Scalable, Privacy-preserving Remote Attestation in and through Federated Identity Management Frameworks

Tamleek Ali\textsuperscript{1}, Mohammad Nauman\textsuperscript{2*}, Muhammad Amin\textsuperscript{2*}, Masoom Alam\textsuperscript{1}

\textsuperscript{1} Institute of Management Sciences, Peshawar, Pakistan
{tamleek,masoom}@imsciences.edu.pk

\textsuperscript{2} City University of Science and Information Technology, Peshawar, Pakistan
{recluze,clickforamin}@gmail.com

Abstract—Creating trustworthy online computing is an important open issue in security research. Trusted Computing aims to address this problem through the use of remote attestation but comes with its own baggage in the form of privacy concerns. Federated Identity Management Systems (FIDMSs), on the other hand, provide another form of trust but lack the ability to measure the integrity of platforms that they vouch for. We note that these two security architectures have reciprocal strengths and weaknesses and can be combined to create an architecture that addresses the concerns of both. In this paper, we propose an extended FIDMS in which the identity provider not only vouches for the identity of a user but also for her platform’s integrity. In this way, we (a) allow a service provider to establish trust on a client platform’s integrity without sacrificing privacy; and (b) create a feasible and scalable architecture for remote attestation. We describe our proposed architecture in the context of Shibboleth FIDMS and provide the details of the implementation of this system.

I. INTRODUCTION

Security is an ever growing concern among enterprises throughout the world. With the increase in number of online businesses and individuals relying on online services, these concerns have grown exponentially. Examples of such scenarios are resource sharing, Internet banking and business-to-business scenarios. Different facets of security in such scenarios have surfaced in the past such as identity management, privacy, authorization, authentication, and integrity. Most of the solutions aimed at addressing these security concerns are disconnected and are governed by different security models addressing one (or only a few) individual concerns. For example, several models exist for authorization that are highly expressive when it comes to policy specification but do not cater to integrity requirements. Similarly, integrity models such as Biba\textsuperscript{4} or Clark-Wilson\textsuperscript{6} do not try to address concerns of identity management. Due to this limited scope of each model, one aspect might be resolved but another missed. In this contribution, our focus is on creating an architecture for combining Federated Identity Management Systems (FIDMS) with integrity measurement models in order to create a framework capable of providing authentication, authorization and attestation in a single package.

FIDMSs provide mechanisms for authentication, authorization and privacy preservation of identities of individuals. In FIDM scenarios, a user requests a service or resource from a Service Provider (SP). It is assumed that the SP and the user do not have a prior trust relationship between each other and that the SP requires some authentication information to make access decisions. Each user may be associated with one or more identity provider (IdP). The user trusts these IdPs and the IdPs are able to authenticate the users and provide credentials associated with the users to the SPs based on the privacy settings of the users. These SPs make decisions to allow or deny access based on these credentials and their own policies. The gist of the protocol is that 1) users trust the IdP with their credentials, 2) the IdP caters to the privacy requirements of its users and 3) the SP trusts the IdP to provide the credentials of the user without requiring the specifics of who the user actually is.

In this way, FIDMSs allow the delegation of authentication to an entity which is more acceptable to the user for her authentication. Similarly, each SP does not need to have knowledge of an infeasibly large number of users that may lead to redundancy and lack of scalability as the number of SPs increases. Thus, FIDMS cater to the problems of authorization, authentication and privacy. However, the problem remains that while the user requesting data or services from the SP might be authentic and trusted not to misuse the resources provided, her platform might not be in a trusted state.

* At the time of writing of this paper, these authors were placed at Institute of Management Sciences.
This lack of trust in the integrity of the platform might lead to security risks for sensitive resources. We note that remote attestation – an important concept proposed by Trusted Computing Group [19] – is a novel paradigm capable of measuring and reporting the integrity of the client platform in such scenarios. This measurement and reporting allows a challenger to ensure that no malicious software is executing on a target platform or that the target platform will behave in an expected manner [3].

Existing remote attestation techniques propose simple to complex mechanisms of a challenger platform performing attestation of all involved target platforms. While these architectures allow for a simple attestation mechanism, they lack in one aspect that has always been a source of concern among the scientific community. This is the problem of privacy in remote attestation scenarios. Whenever a challenger platform attests a target platform, the target platform must provide detailed information to the challenger regarding its hardware and software configurations. This not only leads to privacy concerns but may also give rise to security risks such as denial of service attacks [14].

In this contribution, we argue that the problems in FIDMS and remote attestation are reciprocal: FIDMS cater to the privacy and scalability requirements of security mechanisms but do not allow for the measurement of integrity of platforms. Remote attestation on the other hand allows for the measurement of integrity but leads to severe privacy threats and scalability issues. We propose the coupling of these two mechanisms for addressing the problems of both through their combined set of strengths. In our architecture, the IdP is made responsible, not only for providing client’s authentication information but also integrity information of the client’s platform. This way, the IdP becomes the challenger and the client only releases her platform’s configuration information to her own IdP – the entity that it already has a strong trust relationship with. The SP, on the other hand, trusts the IdP to provide correct integrity information without having to perform the attestation itself.

We describe the design and implementation of a Trusted Federated Identity Management System that combines the strengths of FIDMS and remote attestation to allow the measurement of the integrity of the client’s platform. This leads to a privacy-preserving and scalable implementation of remote attestation, which can be deployed in a vast majority of FIDMSs operating in the world today.

**Contributions:** Our contributions in this paper are as follows:

1) We remedy the problem of privacy in remote attestation by leveraging a third party between the SP and the client such that the client only has to release her platform’s configuration to a third party with which it already has a strong trust relationship.

2) We incorporate the concept of attestation in federated identity management frameworks so that not only the identity of the user can be verified but also the integrity of her platform.

3) We provide an implementation of our idea in the form of a plugin for the Shibboleth [15] framework – a prominent and widely used federated identity management framework.

**Outline:** The rest of the paper is organized as follows: In Section II we provide some background information about Trusted Computing and FIDMSs. In Section III we detail the proposed architecture for incorporating remote attestation in FIDMSs and the production-level implementation of our proposed extension, which can be deployed in existing FIDMSs. We conclude the paper in Section IV and present future directions in this line of research.

## II. Background

### A. Trusted Computing

Trusted computing [19] is an initiative by the Trusted Computing Group (TCG) [19] for bringing trust to different aspects of computing including the PC client, mobile platforms and storage media. On PC clients, the concept of trust is provided through the use of a hardware chip called the Trusted Platform Module (TPM). TPM is a crypto co-processor that provides several security functionalities such as random number generation, asymmetric key generation and shielded memory locations called Platform Configuration Registers (PCRs). Each PCR can store the configurations of a platform in the form of SHA-1 hashes of different entities such as the BIOS, boot loader, kernel and application executables. Whenever a value has to be stored in the PCR, its hash is appended with the existing value of the PCR and SHA-1 of the resulting structure is stored in the same PCR. This operation is known is *PCR extend* and is the only way of changing the value of PCR (other than a platform reset). This technique, coupled with the irreversibility of SHA-1, ensures that an infinite number of configurations can be saved in a single PCR and that the values of PCRs are tamper-resistant. The aggregate of these measurements can be submitted to a challenging party as proof of a trustworthy configuration of the platform to which the TPM belongs. Trust is established on the value of the aggregate by having it signed with an *Attestation Identity Key (AIK)* that is accessible only to the TPM. The TPM never releases the private part of the AIK for use outside the TPM and a value signed by the AIK thus provides assurance that the aggregate is vouched for by a genuine hardware TPM. Since this operation allows a challenging...
party to verify the trustworthiness of a remote platform, it is known as remote attestation.

B. Remote Attestation

Several techniques of remote attestation build on this basic concept of load-time measurement. TCG's basic proposal is limited to measuring the software systems loaded before the operating system i.e. BIOS, boot loader etc. Integrity Measurement Architecture (IMA) \[12\] extends this mechanism to within the operating system by measuring all libraries and executables loaded during and after the Linux operating system’s boot process. It records the hashes of applications at load-time and maintains a Stored Measurement Log (SML) in the Linux securityfs. During attestation, the SML and the values of the PCR are reported to the challenger, who can then analyze the host of loaded applications and decide whether the platform is in a trustworthy state. For details of this novel concept of integrity measurement, reporting and verification, we refer the reader to \[12\]. The primary concern in using IMA in production environments is that of privacy. Since IMA proposes to release complete information about a platform’s binaries, it makes the platform vulnerable to several attacks such as denial of service etc.

It is important to note that none of the existing remote attestation techniques completely address the issues of privacy. In fact, it seems that remote attestation and privacy are mutually exclusive. Moreover, scalability surfaces as an important limitation in most of these approaches due to the large number and diversity of target platforms. In order to address these concerns, we propose the use of existing identity management frameworks to leverage the benefits of remote attestation while preserving privacy of target platforms and keeping the resulting architecture scalable to large federations.

C. Federated Identity Management Systems

The Internet has enabled organizations to make their data available online through the interface of the web. Secure dissemination of this data to intended recipients only is an important problem that is usually solved through the use of authentication and authorization of users based on username/password pairs. However, in scenarios where several organizations are interconnected, this simple solution no longer remains viable. Each organization has its own sets of username/password pairs and user meta data. Sharing all this information is not only cumbersome but also often impossible due to privacy concerns. Federated Identity Management Systems (FIDMSs) provide a solution to this problem by partially releasing user attributes in an inter-organization scenario. Each organization may have one or more Service Providers (SPs) that provide services and resources to clients across the federation. Moreover, each organization can have an Identity Provider (IdP) that provides meta data about users associated with its respective organization to authorized SPs. In this way, the IdP of an organization protects the privacy of its own users and SPs can make authorization decisions based on anonymized attributes of requesting users. For example, an SP in University of Faraway may decide to release an instructor’s manual to a user because the IdP of University of Homeland has verified that the user is a faculty member (without revealing her name). This coupling of anonymity with authorization has led to wide adoption of FIDMSs over the past few years.

Several production level implementations of FIDMSs exist today among which Shibboleth \[15\] and Liberty Alliance \[12\] are the most popular. In this paper, we focus on Shibboleth due to its modular design and open source nature. Shibboleth is provided as a set of servlets and modules ideally deployed on Apache httpd and/or Tomcat servers. Its main components are the Service Provider (SP), the Identity Provider (IdP) and the Discovery Service (DS). The meta data of SPs and IdPs is stored in the form of XML documents and includes Attribute Release Policies (ARPs), different URLs for redirection among the entities and public keys of different systems in the federation. The SP is primarily composed of the shibd daemon and an Apache httpd module that listens for authorization requests from the web server. The IdP is implemented as a Java servlet and is built on the Model-View-Controller paradigm of the Spring Framework \[16\]. This provides a pluggable architecture for easy integration of extensions to the IdP without having to change the existing code. In the following sections, we describe how we have leveraged this flexibility for incorporating remote attestation in IdPs for providing attestation results to SPs in an FIDMS using Shibboleth.

III. CLIENT ATTESTATION IN FEDERATED IDENTITY MANAGEMENT

The Shibboleth \[15\] project provides a completely open source implementation of a federated identity management framework. The scalable architecture of the Shibboleth IdP coupled with its modular structure and object oriented design in the Java language makes it suitable for our target architecture. The Shibboleth IdP is based on the popular Spring framework which provides easy to use extension points through XML-based configuration files. In this section, we discuss in detail how remote attestation can be incorporated in the IdP of the Shibboleth project using these extension points.
The four main entities involved in the Shibboleth architecture are the Identity Provider (IdP), the Service Provider (SP), the Discovery Service (DS) and the client. In a typical login scenario in Shibboleth, the client requests the SP for a resource. If the resource is a protected one, the SP redirects the client to the DS, which presents the client with an interface to select their preferred IdP.

Once the client makes her decision, she is redirected to the single sign-on authentication system of the selected IdP. Assuming that the IdP employs a username/password authentication mechanism, the client is presented with a login screen where she enters her username and password. If the information is correct, the IdP creates a session for this client and creates a handle, which is communicated to the client’s browser. The browser presents this handle to the SP. The SP can then use the handle to request the client’s attributes from the IdP.

During attribute lookup, the SP sends the session handler to the IdP along with the attribute request using SAML protocol [13]. The Attribute Resolver in the IdP looks up the attributes of the client, verifies that the privacy policy – called Attribute Release Policy (ARP) – authorizes the release of the particular attributes to the SP and releases the attributes in the form of a SAML response. The SP maps these released attributes to environment variables, which can then be read by the different applications. Depending on these attributes, the application may decide to release the secured resource or customize the output for the particular client.

Figure 1 shows the detailed steps involved in our target architecture. Below, we describe how each of the existing modules in the Shibboleth framework can be modified or extended for incorporating remote attestation.

A. Identity Provider (IdP)

The IdP is the primary target of the changes required for remotely attesting the client. In the existing Shibboleth architecture, the IdP is responsible for: 1) establishing the identity of the user and 2) providing the Service Provider (SP) with attributes of the user.

We note that, in this scenario, the SP trusts the IdP completely and bases its access decisions on the information provided by the IdP. The IdP is capable of performing authentication but cannot vouch for the integrity of the user’s platform. On the other hand, the SP may want assurance regarding the trustworthiness of the client platform before releasing sensitive data or objects to it. A naïve approach would be for the SP to perform attestation of the client directly. However, this approach has two problems:

1) In a federation, the SP has to serve a large variety of clients – each with its own set of binaries, hardware and software configurations. Enabling the SP to attest all variants of these is not feasible.
2) The SP is usually outside the control of the organization to which the client belongs. Revealing system properties required for attestation to an untrusted party is not usually desirable.

It is for these two reasons, that we believe the IdP is most suited to act as the attestor:

1) The IdP is specialized for an organization and is already aware of the different users and (in most cases) their platforms. It is therefore more capable of performing their attestation.
2) The IdP is also usually hosted within the organization of the user and is therefore most likely to be trusted by the user. The clients, who are being authenticated by the IdP, are thus likely to
be willing to release information regarding their platform configuration to the IdP.

Thus, using the IdP as the challenger assures that remote attestation is both scalable and preserves the privacy of the users. Below, we describe the details of how we have extended the Shibboleth IdP for performing attestation of clients after successful authentication.

In the officially supported implementation of the Shibboleth framework, the IdP is implemented as a Java servlet built on top of the Spring Framework. Different components of the IdP are described and linked together using XML data structures. Whenever an IdP receives a request for a user credentials, it looks up the XML configuration to decide how that attribute might be resolved. In any case, the user is first authenticated (usually through LDAP or an RDBMS). Afterwards, different Attribute Resolvers are used for retrieving different attributes of the authenticated user. Attribute Resolvers call Data Connectors, which act as a conduit between the resolvers and the data stores. It is the responsibility of the connectors to communicate with the data stores in a language understandable by the data store and to map the returned attribute values to a structure parseable by the resolvers. In our architecture, we have developed an IntegrityResolver that communicates with the IntegrityProviderDataConnector. This connector populates the PlatformIntegrity attribute requested by the IntegrityResolver. Essentially, the IntegrityProviderDataConnector is responsible for calling a Validation Service (VS – cf. Section II), passing the attestation type and the client’s IP address to the VS and receiving the attestation result. The result is then returned as the value of the PlatformIntegrity attribute to the IntegrityResolver. Figure 2 shows the relevant portion of code of the custom data connector.

1For the sake of brevity, we omit the detailed steps involved in the protocol and focus on the relevant portions in the following discussion.

Using the standard resolver and connector mechanism of the IdP, we can easily fine-tune the different parameters of our attestation framework such as the VS to be used, the length of time to cache the attestation results, the attestation technique to be used etc. All these configurations can be made at runtime by changing the XML configuration file (attribute-resolver.xml) at the IdP. Moreover, the Attribute Release Policy can be used to restrict this attribute so that only a subset of SPs will be provided with the results of attestation based on the organizational policy. The actual attestation is taken care of at the VS which may or may not be running on the same platform as the IdP. Since the operation of the VS depends on the trust tokens returned by the client, we first describe the details of the client platform in our architecture.

B. Client

The target of remote attestation in our scenario is the client platform. According to TCG’s specification [11], Trusted Computing is completely opt-in – no remote attestation can take place unless the client has enabled and activated the TPM. In accordance with the spirit of this specification, our proposed architecture is also opt-in. For a client to be attested, the user (or an administrator) must install a remote attestation aware browser for handling remote attestation requests. Below, we describe the architecture of this browser and other system requirements for the client platform.

1) Trusted Computing Prerequisites: Our architecture assumes that the client platform is a TPM-enabled system with the following prerequisites:

1) The client must have the TPM enabled, activated and owned. The owner can be the user currently being authenticated or the administrator of the system.
2) The client (or an administrator) must have created an Attestation Identity Key (AIK) (cf. Section II-A and registered it with a PrivacyCA [2]. (Since this AIK will only be utilized within a specific domain by a single IdP, this step can be omitted if the AIK is reported to the IdP after creation.)
3) The authorization secret of the AIK must be accessible to the browser application for performing quotes over the PCRs.

Each of these prerequisites requires a one-time operation only. To increase the usability of our approach, we embed the recurring operations within the browser in the form of an attestation daemon. This attestation daemon must be able to:

1) respond to the attestation requests from the IdP,
2) request the TPM to perform a quote over the PCRs, which store the event logs recorded by Integrity Measurement Architecture,  
3) read the Stored Measurement Log (SML) and report it during attestation and  
4) process and create SAML requests and responses for attestation.

In order to cater to all these requirements, the attestation daemon forms part of a larger framework termed as BAClient. BAClient is the behavioral attestation client that is embedded in the browser for performing the above mentioned functions. During attestation, the attestation daemon calls the different attestors for performing different operations. PCRAttestor takes the attestation challenge sent as part of the attestation request and sends the challenge to the TPM for performing quote over PCR-10. This PCR is populated by IMA (cf. Section II-B). The quote is encoded in an XML node to be embedded in the SAML response. The attestation daemon also calls the SMLAttestor. This attestor reads the SML from the securityfs and encodes it in an XML NodeSet. The architecture proposed in this contribution is pluggable. If, in the future, the IdP decides to implement a different attestation technique, it may require BAClient to implement other attestors too.

Once these trust tokens are collected by the different attestors, the attestation daemon encodes them in a single SAML response and returns the resulting assertion as the attestation response.

We have incorporated these changes in the open source Java-based browser Lobo [18] and have utilized this browser as the client in our architecture. The customized Lobo variant – termed Trusted Lobo – implements the BAClient module that can respond to attestation requests from a challenger. Figure [3] shows the details of this module as implemented in Trusted Lobo.

These trust tokens, encoded as a SAML response, are returned to the VS that verifies them to ensure that the platform is in a trustworthy state.

C. Validation Service (VS)

The VS acts as the primary challenging party in our target architecture. It can be a part of the IdP platform or a dedicated entity within the organization to which the user belongs. For the purposes of our discussion, we treat the VS and the IdP as separate, independent entities. The IntegrityProviderDataConnector at the IdP sends a request for performing attestation to the VS. The Challenger at the VS (cf. Figure [4]) creates an attestation request and sends it to the client. This request includes a random nonce to ensure that the attestation response is fresh – thus catering to several attacks such as the replay attack. The attestation response is generated at the client as discussed in the previous section. When the VS receives this response, it calls all concrete instances of AbstractVerifier subclasses to ensure that all of them can successfully verify that the returned response is trustworthy. Each verifier looks at a different aspect of the response. This modular nature of the verifiers ensures scalability of the approach by allowing the introduction of more verifiers in the future to accommodate new attestation techniques. In our current scenario, we have two verifiers:

1) **PCRVerifier**: verifies the digital signatures on the response to ensure that the PCR quote is signed by a genuine hardware TPM. Since the digital signature also includes the nonce, freshness of the response is also assured if the digital signature is validated successfully.

2) **SMLVerifier**: iterates over all the entries in the SML and performs two tasks:

   a) Checks to make sure that the hash of the loaded executable is a known good one. For this purpose, the VS maintains a database of known good hashes. This is a tedious job and is one of the major bottlenecks in using attestation in heterogeneous environments. In our scenario, however, the VS is only responsible for performing attestation of clients within the organization and as such is required to keep track of a limited number of
possible configurations. This greatly reduces the complexity of the problem as platforms within an organization are likely to have a smaller configuration space.

b) After successfully verifying all entries, the SMLVerifier computes the expected PCR aggregate using the formula:

\[ PCR_{10}^{i} = SHA^{-1}(PCR_{10}^{i-2} || SHA^{-1}(e_{i} - 2)) \]

where \( PCR_{10}^{i} \) is the value of the PCR-10 after entry \( i \), \( e_{i} \) is the \( i^{th} \) entry in the SML, \( i \) ranges over \( 1 \ldots n \) and \( n \) is the number of entries in the SML. If the value of \( PCR_{10}^{i} \) matches the value of the PCR signed by the client’s TPM, SML verification succeeds.

The combined result of all verifiers is communicated to the IdP which is then passed on to the IntegrityResolver by the IntegrityProviderDataConnector. Figure 5 shows the log created during successful attestation. The IdP inserts the PlatformIntegrity attribute in the set of user’s credentials. The release of this credential to the requesting SP depends on the Attribute Release Policy mentioned earlier. After the SP receives this credential, it may make access decisions based on the integrity status of the client platform.

D. Service Provider and Discovery Service

In our proposed architecture, the Service Provider (SP) is only extended in a limited way. We have changed only the application end of the SP, leaving the module (mod_shib) and other aspects of the Shibboleth SP intact. We have created a sample application, which protects a subset of all files hosted by the web server. The decision to require attestation of the client platform before releasing a sensitive resource is solely up to this application. In Shibboleth, all the attributes returned by the IdP are made available as environment variables. Assuming that the IdP’s Attribute Release Policy mandates the release of PlatformIntegrity attribute to the SP, this variable will be made available to the challenging application. The application can then make the decision to release the resource based on the owner’s policy and the attestation result. The description and specification of this policy can be part of a larger authorization model and is outside the scope of this paper.

The final component of the Shibboleth architecture is the Discovery Service (DS) that used to present the user with an interface that allows the selection of an IdP. Keeping to our principle of minimum required change to the Shibboleth architecture, we have made no effort at changing this behavior of the DS.

IV. CONCLUSIONS AND FUTURE WORK

Identity management and remote attestation are two important aspects of trust in inter-organizational scenarios. Identity management enables privacy preserving release of credentials based on an organizational policy but is unable to provide any details about the integrity of a client’s platform. Remote attestation can cater to this problem but is sorely lacking when it comes to preserving privacy of individuals. In this paper, we have
proposed a framework that combines the best of both worlds by enabling the IdP to vouch for the integrity of the client that it is authenticating. Since the IdP is already trusted by SPs to provide authentication results, it can also be trusted to perform attestation and make anonymized results available to SPs. By delegating the process of attestation to an organization’s own IdP, the process is made scalable as well as privacy preserving. We have developed a prototype implementation of our technique in the context of the popular Shibboleth framework and have provided details of implementation and design decisions of our framework. In this way, we provide a complete set of components of our architecture that can be used at the client and IdP ends to report the integrity of a client platform in a trusted manner. We believe that this framework can be used to solve two main problem faced by remote attestation today: scalability and privacy.

Currently, we have incorporated IMA as the remote attestation technique but have pointed out that this is purely for the purposes of this paper. The framework itself is pluggable in that it can incorporate any attestation technique as deemed fit by the IdP. An important future direction for research in this area is to incorporate more sophisticated techniques such as PRIMA [9], property-based attestation [11], program execution [8] or model-based behavioral attestation [3][10] in this framework.

REFERENCES


