Scalable Protocol for Remote Integrity Attestation of Cloud Based Distributed Services

Siamak Azadiabad
Department of Computer Engineering and Information Technology
Amirkabir University of Technology
Tehran, Iran
sazadi@aut.ac.ir

Hossein Pedram
Department of Computer Engineering and Information Technology
Amirkabir University of Technology
Tehran, Iran
pedram@aut.ac.ir

Mohammad Reza Abbasy
Advanced Informatics School
University Technology Malaysia
KL, Malaysia
ramohammad2@live.utm.my
mra@computer.org

***Abstract*— although, using cloud-based services is increasing nowadays, but for many sensitive applications, users need to ensure that the processes are performing correctly in the cloud. In this paper, we propose a scalable method for remote attestation of distributed services integrity in the cloud, in comparison with other similar works. This method allows performing attestation at any time, and it is easy to implement. Users also can identify malicious cloud systems in order to report to the cloud service provider in most of the cases.**

***Index Terms*—Cloud Computing, Distributed Cloud Service, Integrity Attestation, Cryptography, Digital Signature, Endorsement.**

I. INTRODUCTION

Nowadays, Cloud Computing is becoming a general platform for computing; since after developing some technologies such as Internet and virtualization, the possibility of presenting services by cloud computing has become an easy solution [1].

Cloud Computing is based on three service models mainly, which one of them is Software as a Service (SaaS). In the SaaS model, a cloud service provider is responsible for managing and hosting service applications in its servers. Thus, dealers of different services can host their applications in the cloud and pay for its charge. Then, each dealer can sell its services to the customers to have revenue [2]. The major reason for popularity of this model is lower costs of services because, in SaaS model, we have better utilization of resources by applying resource sharing. So, this ability of increased utilization will increase the final profit [1].

If users decide to use these services, first of all they need to be sure about the security of the services in different aspects. Cloud computing inherits many security issues from classic computing. However, in cloud computing, because of its special characteristics, some security problems such as integrity attestation are more significant [3]. The confidence of the integrity of service is one of the main security issues in the SaaS model. The users must be able to ensure about the validity of the process results which they receive from the cloud [4], especially for critical data processing applications. There are several papers in service integrity, which has its own advantages and disadvantages. In this paper, the goal is, presenting a scalable solution without any extra hardware or server in the cloud and just by extending some abilities to the service applications. In our architecture, the end user can ensure about the integrity of processes remotely and also it will be able to detect and report malicious servers most of the time.

The remainder of this paper is organized as follows: first of all, some related works are presented. After that, in section IV as a technical core of this paper, the proposed method including assumptions, our contribution, security analysis, and performance and scalability is discussed. Finally, this paper is concluded in section V.

II. RELATED WORKS

Many techniques for remote authentication and attestation utilize challenge and response procedures to ensure that programs are run correctly. Also, system-level verification can be performed locally [5] which are done by a trusted hardware or additional kernel in the system. In such methods, a secure entity must exist in the server in order to examine the verification of the other entities. These methods are not scalable because the cloud provider must utilize additional hardware or software. For example, BIND has provided a method for examining verification of the program execution [6] which is based on this approach. But, in a SaaS model, which clients or service dealers do not have administrative access to the cloud servers, the assumption of trusted entity existence in the servers that can support large different number of services, would not be easily acceptable. As another solution for process integrity check, in the works from DANCE Research Laboratory at the University of North Carolina, they have researched on a way to perform remote service integrity attestation [7, 8, 9, and 10]. They proposed to have a trusted node in the cloud that is the gateway of sending and receiving users' request and response. They also have mentioned an algorithm in order to detect the unhealthy nodes. One drawback of their solution is that this trusted node will cause some problems such as bottleneck. The other problem is that if we can trust to a node in the cloud, why all the servers cannot be trusted? The algorithm presented in those papers is approximately same as each other and has been presented for the first time in [7]. Note that, their contribution is based on the creation and analysis of a complete graph and solving a maximum clique which is an NP-complete problem [7]. In the other three papers, they tried to improve the time complexity of this solution by heuristic methods.

In the paper [11], a solution based on analysis of trust networks in the cloud in order to identify unreliable colonies has been provided. This approach has an acceptable scalability. But, it lacks uncertainty in decision-making about nodes trustworthiness.

In some other articles, the verification of the localized applications in the cloud is taken into consideration. As an example, in the paper [12], they applied a verification method for the variables in the RAM. In this work, in order to solve the problem of the integrity verification at the run time of the process, they define specific attributes for the variables, so that, by monitoring or controlling these attributes, unwanted changes to variables can be detected. The main innovation in this paper is introducing "scoped invariants" as an integrity property. These scoped invariants are codes or variables which under terms and conditions (scope) are constant. Therefore, under these circumstances, any changes in scoped invariants equal to losing their verification. In this paper, it is assumed that there is administrative access to the cloud server.

In the paper [4], sealing the instances of an executed program is their solution. So the cloud server creates an instance of an executed program. Then cloud issues a X.509 certificate for it and signs the instance by the private key. Finally, the user can be ensured about the integrity of the process by verifying the signature. The problem is that in this method, cloud server is trusted.

III. Service and Attack Model

A. Service Architecture

As mentioned before, in a SaaS model, service resources are communally used. It means that, several multiple service dealers can run their application in the same cloud space. Therefore, cloud service provider must protect each application from another one. On the other hand, each service dealer can host its application in more than one server in the cloud space. For example, a web service dealer may want to balance requests among the different replicas in the cloud by DNS Load Balancing techniques [13]. So, identical copies of the application would be distributed over multiple servers in the cloud.

Clients are on the other side that utilizes the cloud based services. In the previous example, a client may specify a particular server by determining its IP address or may send its request to a server determined by DNS load balancing mechanism [13].

Distributed service architecture in the cloud can be seen in Fig.1. In this architecture, the **k** numbers of client utilize different cloud based services. There are **n** numbers of server in the cloud. Also on any server, a number of applications from different service dealers can be hosted [13].

The same application may be distributed between these **n** numbers of server. This is why this model is called the distributed service architecture. In this architecture, clients who have purchased a specific service from a service dealer are able to have access to this service or application on all servers. Besides, it is possible to apply PKI for some applications such as non-repudiation and digital signature in this architecture.

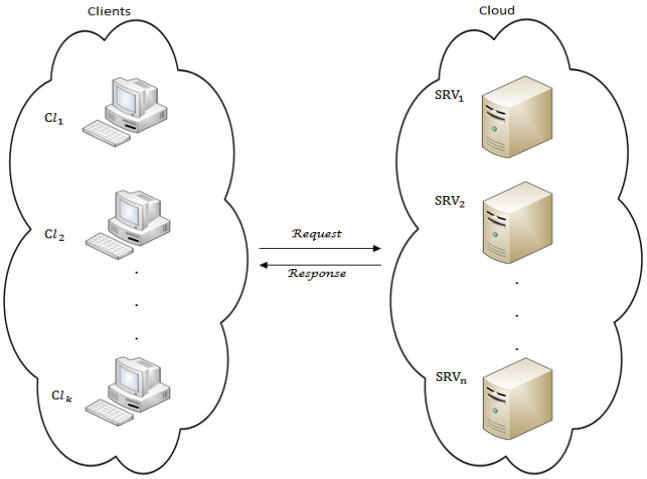

Fig. 1. Architecture of distributed cloud service

B. Attack Model

Programs and applications on the cloud servers may be manipulated for various reasons. It is also possible which a software or application be compromised, or even occurrence of a defect or malfunction can cause unwanted changes to them. As a result, losing the integrity of the cloud processes will also be possible. On the other hand, unwanted or unauthorized changes on applications may not be easy to discover by the cloud provider. For example, an attacker can gain unauthorized access to the server, and he or she can compromise operating system by kernel level rootkits which this is very difficult to detect [14]. If unauthorized or unwanted changes cause program failures, the user will not receive the service, and he or she can make a report to the vendor of service. But, on the other side, manipulation in an application may cause stealthy changes in the output of the program. The changes that are not easily detectable are stealthy. In this case, process shows incorrect results intelligently, and the user will not be able to detect it. For example, consider a user receive a service from a cloud that it is a processing of massive statistical data and presenting an average as a result. So, in this scenario a few changes in the average value cannot be detected by the end user. It should be noted that the end users are often using poor processing systems such as a PC and it is not possible to re-compute those bulky calculations.

IV. Proposed Solution

Considering the distributed service architecture in Fig. 1 and the attack model, the goal of this paper is taking advantage of similar program or application distribution on multiple cloud servers in order to achieve the remote integrity attestation of processes possibility by the end users. Thus, the users should be able to distinguish untrusted process results from trusted ones in our solution. In the present work, beside security and scalability of the solution, there is no need to use any additional hardware or trusted node in the cloud.

A. Assumptions

First of all, in the proposed solution, the user should not need administrative access to the servers in the cloud. This assumption is completely compatible with the SaaS model. Therefore, users may not have privileges to install and run the traditional Integrity Checker programs on the cloud server. Therefore, from users' point of view, the cloud servers are black-box machine. The next assumption is that the user cannot trust to what the cloud administration claims about the integrity of the program. Even the cloud service provider was trustworthy; the cloud administration may not be able to detect unauthorized changes or any compromise by an insider in the cloud. The third assumption is that the number of healthy applications is more than the number of manipulated applications. Thus, healthy applications are the majority population of the distributed service. This assumption also has been considered in other models [7, 8, 9, 10 and 15]. Consequently, if we consider **n** distributed copies of a program, at least **[n/2]+1** copies will be in normal behavior. But, we do not know exactly which one is healthy and which one is not. The final assumption is that if the same request and input are given to the healthy distributed copies of an application, all the results will be identical. This assumption for applications that are stateless and deterministic is reasonable [8]. However, for the applications that are state-dependent, it is possible to transfer them to stateless conditions. This issue has been discussed in [10].

B. Our Contribution

- Aims

As mentioned before, from a user's perspective, in distributed service model, there are a number of servers in the cloud, which offer services in a black-box manner. In this study, we call this server a *Node*. The goal of our proposed scheme is providing a way to identify incorrect results from healthy ones, with the following objectives:

(1) No need to additional trusted node in the cloud
(2) Scalability
(3) Definite attestation result, so any user will be able to ensure about the results correctness
(4) Possible to perform check, any time that a user prefers
(5) Detecting healthy nodes from unhealthy nodes if it is possible

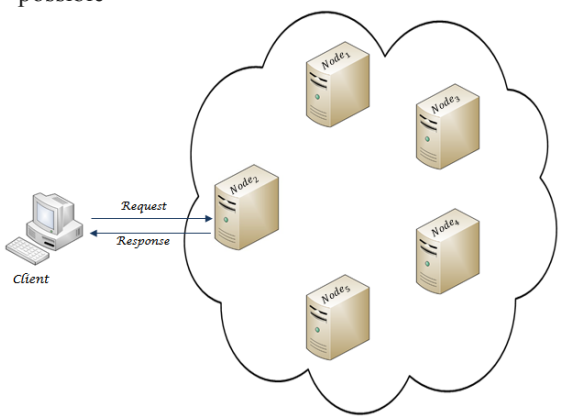

Fig. 2. Distributed service usage model

- Service Model

Fig. 2 demonstrates a basic distributed service usage model with five nodes, and how a user makes contact with one of them in the cloud environment. In this figure, these five nodes provide a distributed service to the legal users. For example, a request is submitted by a user to the *Node*$_2$, and after performing the required processes by the application in the *Node*$_2$, corresponding response gets back to the user. This node can be chosen randomly or based on an algorithm.

- Attestation Protocol

In order to detect potential tampering or forging of messages, users utilize digital signature by Public Key Infrastructure (PKI) enabled applications. Each user in the client side and application in the server side has a public and private key with required cryptographic functions for the purpose of applying PKI as needed. Whenever a user decides to attest the integrity of the process result, it can ask from the recent node to prove its process results correctness. The attestation protocol steps are shown in Fig.3 which is explained as follow:

(1) First, the *Client* sends the message I, which it is a service request. This service request will determine what process the distributed application should do, and it contains all the inputs that the application needs. We call this *Request*.
(2) Then, the *Client* receives the message II, which it contains the result of the process. We call this *Response*. Up to this point, it is not possible for *Node*$_2$ to know that Client may want to do attestation. So, it processes the *Request* and sends the *Response* as usual. If the service of *Node*$_2$ behaves healthy, the correct result will be delivered, and if the integrity of service has been damaged, it may deliver an invalid result.
(3) Now, if the *Client* decides to do attestation, it asks *Node*$_2$ by message III to prove its *Response* validity. This message contains recent *Request* and *Response* which has been signed by *Client* private key. We call this *Proof*.
(4) Then, *Node*$_2$ must send a request to the other nodes by message IV and gathers at least **[n/2]** confirmations. After that, *Node*$_2$ must send those confirmations to the Client. In the Fig. 3, **n** is the number of the distributed applications or the number of the nodes. In other words, *Node*$_2$ must send the message III to the remaining nodes in the cloud and ask them to respond an *Endorsement* message. If *Node*$_2$ does not send the message IV, it will not receive any *Endorsement* from healthy nodes. Also, by considering digital signature in *Proof*, if a manipulation happens in *Proof* messages, it is possible to detect. So, *Node*$_2$ has no way, but sending original *Proof* request to the other nodes without alteration.
(5) In this protocol, unhealthy nodes may have one of the malicious or healthy behaviors. But the healthy nodes

(1) are in the majority; (2) always produce the correct and identical response for the same request; (3) and run integrity check protocol correctly and without malfunctioning. Thus, in this step, these nodes extract the *Proof* request in IV. Then, they perform the same process on *Request* and produce their own *Response*i and compare this result with the original *Response*. After that, an *Endorsement* message is produced by each node and if *Response* and *Response*i are consistent, this node put *OK* into the *endor* field of *Endorsement*. Otherwise, the value in this field will be *NOK*. Now, each node signs the *Endorsement* message by its private key and sends it to the *Node2* by message V.

(6) Then *Node2* should send all the *Endorsements* to the *Client* by message VI.

(7) When *Client* receives the *Endorsements*, it will check the signature of them, in order to eliminate altered or duplicate messages. Finally, the numbers of *OK* are counted by the *Client*. If this is greater than or equal to **[n/2]**, the result of *Node2* process is correct. Otherwise, *Node2* is an unhealthy node and the *Response* is incorrect. So, *Client* must put this node to a black-list and make a report to the cloud service provider for cleaning this node.

(8) Whenever the cloud service provider reports the health of the *Node2* to the *Client*, then user should remove this node from its black-list.

For the purpose of clarifying details of the messages, all the notations of the protocol have been mentioned in table 1:

The details of the message exchange for the Fig. 3 are:

TABLE 1: NOTATIONS

Notation	Meaning
$Client_i$	Client number i
$Node_i$	Node number i
$Client_{i,ID}$	ID for Client number i
$Node_{i,ID}$	ID for Node number i
$Request$	Client request for a process on a provided input
$Response$	The result of process on *Request*, which would be attested, if the client decide to check its integrity
$Response_i$	The result of process on *Request* by endorsing Node number i, which would be compared with *Response*
$Sign_x(Y)$	Digital Signature for Message *Y*, signed by x
+	Concatenation
$endor_i$	Negative or Positive endorsement from Node number i
ts_i	Time stamp
$A \rightarrow B:C,D$	A sends C and D as a message to B

The details of the message exchange for the Fig. 3 are:

I. $Client_1 \rightarrow Node_2: Request$

II. $Node_2 \rightarrow Client_1: Response$

III. $Client_1 \rightarrow Node_2: Sign_{Client_1}(Request + Response + Client_{1,ID}), Request, Response, Client_{1,ID}$

IV. $Node_2 \rightarrow Node_1: Sign_{Client_1}(Request + Response + Client_{1,ID}), Request, Response, Client_{1,ID}$

$Node_2 \rightarrow Node_3: Sign_{Client_1}(Request + Response + Client_{1,ID}), Request, Response, Client_{1,ID}$

$Node_2 \rightarrow Node_4: Sign_{Client_1}(Request + Response + Client_{1,ID}), Request, Response, Client_{1,ID}$

$Node_2 \rightarrow Node_5: Sign_{Client_1}(Request + Response + Client_{1,ID}), Request, Response, Client_{1,ID}$

V. $Node_1 \rightarrow Node_2: Sign_{Node_1}(Request + Response + endor_1 + Client_{1,ID} + Node_{1,ID} + ts_1), endor_1, Node_{1,ID}, ts_1$

$Node_3 \rightarrow Node_2: Sign_{Node_3}(Request + Response + endor_3 + Client_{1,ID} + Node_{3,ID} + ts_3), endor_3, Node_{3,ID}, ts_3$

$Node_4 \rightarrow Node_2: Sign_{Node_4}(Request + Response + endor_4 + Client_{1,ID} + Node_{4,ID} + ts_4), endor_4, Node_{4,ID}, ts_4$

$Node_5 \rightarrow Node_2: Sign_{Node_5}(Request + Response + endor_5 + Client_{1,ID} + Node_{5,ID} + ts_5), endor_5, Node_{5,ID}, ts_5$

VI. $Node_2 \rightarrow Client_1$ All the received messages in step V

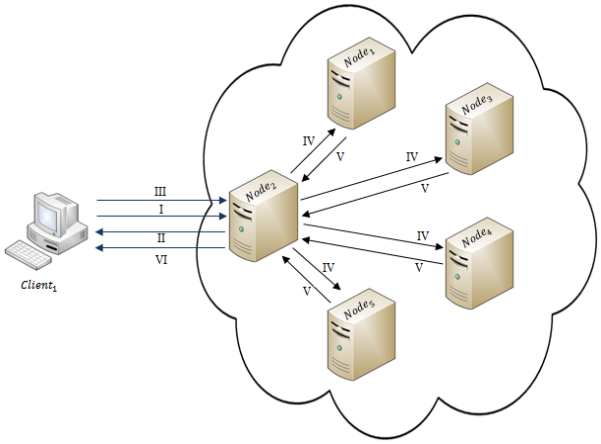

Fig. 3. Sequence of messages in attestation protocol

To stay healthy nodes in the majority from clients' point of view, we propose that as soon as a client detects a malicious node, it should report this to the cloud service provider. So, cloud service provider can notify all the other clients to avoid using this node until the service get cleaned, and secure unhealthy node as soon as possible.

C. Security Analysis

As mentioned before, because of using standard PKI and cryptographic message signing techniques, we can make sure that the message contents in III, IV, and V are not altered, and their authenticity can be proved securely [16]. Also, due to

time stamp, *Endorsements* will not reuse and, their freshness is assured. Moreover, in time of producing endorsement message by each node, colluding with $Node_2$ by unhealthy nodes cannot change the *Client* recognition about the correctness of the *Response*. If we assume the worst case scenario which $Node_2$ has made incorrect result and all other unhealthy nodes have endorsed this node, at least **[n/2]+1** nodes will not approve the *Response* because the majority of nodes are healthy.

D. Performance and Scalability

Performance and scalability of our approach can be explained in the number of sending messages in the network and order of processing time. For any number of nodes, the number of messages I, II, III, and VI are constant. Messages IV and V are sent in the cloud for **(n-1)** times. If we consider the number of message VI equal to the number of messages in V, in order to check the integrity of a *Response*, we have **4+3*(n-1)** messages which the order will be **O(n)**. To verify the integrity of a process in the cloud, each node must process the *Request* one time and also all nodes (except $Node_2$ in our example) produce *Endorsement*. Therefore, we will have **2*n-1** processing time inside of the cloud. Client should also check the signature of **n** messages and the number of *OK*. Thus, **2*n** calculations should be done. So we have totally **4*n-1** processing time in order of **O(n)**.

V. Conclusion and Future Work

In this paper, we presented a scalable method which the client can achieve proof of integrity of data processing. Not only our scheme is applicable on distributed services in a SaaS model with considerable scalability and simplicity in usage, but also it does not need to have additional hardware or trusted node in the Cloud. One of the strength points of our method is the usage of standard cryptographic algorithms in the message signing steps. Therefore, users will be able to check services integrity in an easy and secure way at any time.

It must be mentioned that in this method, it is possible for the malicious node that does not show an unhealthy behavior. So, although we can ensure about the integrity or invalidity of the process output, it is not possible to ensure all the nodes that did their processes correctly, are healthy or not.

References

[1] M. Cusumano, “Cloud computing and SaaS as new computing platforms”, Communications of the ACM, vol. 53, 2010, pp. 27-29.

[2] P.A. Laplante, J. Zhang, J. Voas, "What's in a name? Distinguishing between SaaS and SOA", IT Professional, vol. 10, 2008, pp. 46-50.

[3] S. Subashini and V. Kavitha, “A survey on security issues in service delivery models of cloud computing”, Network and Computer Applications, vol. 34, 2011, pp. 1-11.

[4] A. Brown and J. S. Chase, “Trusted Platform-as-a-Service: A foundation for trustworthy cloud-hosted applications”, 3rd ACM Workshop on Cloud Computing Security, 2011, pp. 15-20.

[5] J. Garay and L. Huelsbergen, “Software integrity protection using timed executable agents”, Proc. ACM Symposium on Information, Computer and Communications Security (ASIACCS), 2006, pp. 189-200.

[6] E. Shi, A. Perrig, and L. V. Doorn, “Bind: A fine-grained attestation service for secure distributed systems”, Proc. IEEE Symposium on Security and Privacy Conf., 2005, pp. 154-168.

[7] J. Du, W. Wei, X. Gu, and T. Yu, “Runtest: Assuring integrity of dataflow processing in cloud computing infrastructures”, ACM Symposium on Information, Computer and Communications Security Conf. (ASIACCS), 2010, pp. 293-304.

[8] J. Du, N. Shah, and X. Gu, “Adaptive data-driven service integrity attestation for multi-tenant cloud systems”, International Workshop on Quality of Service (IWQoS), 2011.

[9] J. Du, D. J. Dean, Y. Tan, X. Gu, T. Yu, “Scalable distributed service integrity attestation for Software-as-a-Service clouds”, IEEE Transactions on Parallel and Distributed Systems Conf., vol. 25, 2013, pp. 730-739.

[10] J. Du, X. Gu, T. Yu, “On verifying stateful dataflow processing services in large-scale cloud systems”, 17th ACM Conference on Computer and Communications Security, 2010, pp. 672-674.

[11] S. Pandit, D. H. Chau, S. Wang, and C. Faloutsos, “NetProbe: A fast and scalable system for fraud detection in online auction networks”, Proc. 16th international Conference on World Wide Web (WWW), 2007, pp. 201-210.

[12] J. Wei, C. Pu, C. V. Rozas, A. Rajan, and F. Zhu, “Modeling the runtime integrity of cloud servers: A scoped invariant perspective”, IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom), 2010, pp. 651-658.

[13] M. Colajanni, V. Cardellini, P. S. Yu “Dynamic load balancing in geographically distributed heterogeneous web servers”, Proc. 18th International Conference on Distributed Computing Systems (ICDCS), 1998, p. 295.

[14]R. Riley, X. Jiang, D. Xu, “Multi-aspect profiling of kernel rootkit behavior”, Proc. 4th ACM European Conference on Computer Systems, 2009, pp. 47-60.

[15] L. Lamport, R. Shostak, and M. Pease, “The byzantine generals problem”, ACM Transactions on Programming Languages and Systems, 1982, pp. 382-401.

[16] R. Rivest, A. Shamir, L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems", Communications of the ACM, vol. 21, 1978, pp. 120-126.