Trusted Anonymous Execution: A Model to Raise Trust in Cloud

Zhexuan Song
Fujitsu Laboratories of America
Email: zhexuan.song@us.fujitsu.com

Jesus Molina
Fujitsu Laboratories of America
Email: jesus.molina@us.fujitsu.com

Christina Strong
University of California, Santa Cruz
Email: crstrong@soe.ucsc.edu

***Abstract*—Software-as-a-service (SaaS) provides developers a new convenient venue to distribute software by utilizing a cloud computing infrastructure. But as vendors start to deploy applications, and users upload data in cloud to utilize them, a new privacy concern arises, because data users would like to preserve their data (and maybe even their identities) private from the software provider. While cloud providers pledge to preserve data privacy, the current SaaS architecture makes it difficult to provide any assurance that the software in the cloud will not be able to make copies or redistribute the data it used.**

In this paper, we propose a new cloud based infrastructure which allows a clean differentiation between applications and data. We further utilize this differentiation to introduce the concept of trusted data binding, enforcing policy usage on application over data sets with the aid of trusted hardware such as the trusted platform module. We implemented our idea in a prototype system deployed in Amazon EC2, where software providers can upload software and data owners can search for algorithms to be executed privately on their data sets, with policy options such as a number of executions, data expiration and deletion, and encryption of data at rest. We believe that our contributions will be very beneficial for fields such as bioinformatics and software validation, were the software is executed against very sensitive data sets and require a high amount of computational resources.

***Keywords*-cloud computing, trusted binding**

I. Introduction

As cloud computing promises limitless computing resources and storage space, the traditional way of purchasing an application and executing the application within a controlled physical environment becomes less attractive. One important reason is that the elastic model of cloud computing allows user to access a large set of resources during the application execution time quickly and inexpensively. In most cases, building such an execution environment for the application to achieve similar performance "locally" could be very expensive in both time and money.

When the cloud computing changes people's view about the way of using computing resources, it also triggers a similar shift in the software delivery model. Software-as-a-service (SaaS) becomes an alternative model and it attracts a lot of attention [1]. The benefits of adopting SaaS include (1) the model saves the overall cost because users only pay when they use the software and the investment on unnecessary software purchases will be decreased; (2) continuously upgrading the software for new features and/or better performance brings extra cost for users in the traditional model, while using SaaS saves such cost; and (3) SaaS model makes the decision to migrate from one software to another a much simpler decision as the initial investment was much lower

However, the SaaS model gives software providers an unprecedented access to data uploaded by users. At execution time the control of the data is handed over from the user (data owner) to the software provider. Furthermore, the results generated from the software execution, in theory, are under the control of the software provider. This raises a new concerns about "trust" [2] on software providers. Certain data, such as medical records, is sensitive by its nature, and handing it over to software providers without having a very high level of trust on them beforehand is definitely unacceptable for users.

Besides software, data is another kind of assets because it takes a lot of efforts to collect and maintain certain data sets. To data providers, cloud, especially storage-as-a-service, promises "limitless" storage space and handles tedious routine management work (e.g. backup), and it is a good replacement for "local" storage. The same trust concern happens when data providers do not want to give out the whole data set to data consumers, and data consumers do not want to share their programs with data providers.

In this paper we introduce mechanisms to separate software from data, so we can create a trusted binding between the software or algorithms and the data set. Our goal is to create a venue for software providers to share algorithms in the cloud, even the ones which are supposed to run over very sensitive data sets. This is especially important for algorithms that require a large amount of resources in fields such as bioinformatics (e.g. predictions of protein structure [3], analysis of mutations in cancer) and software verification [4]. Currently, this type of applications is run locally to prevent both data or algorithm leakage, which prevents the spread of important advances on these fields.

As a prototype we created a cloud based software sharing service. Utilizing the service, software can be uploaded independently of data sets. The owner of a sensitive data set that needs some sort of processing can search or request for suitable software for the task. Both software and data are encrypted while at rest, and accessing the keys triggers a logging mechanisms to ensure that software is utilized only with the allowed data sets.

The outline of the paper is as follows. We begin by defining the requirements of the execution model (section II). We then describe the high-level architecture of our solution (section III). This is followed by an overview of a concrete implementation of our prototype system (section IV). After that, we will further discuss the methods to improve trust assurance by binding application and data with the aid of trusted hardware (section V). Finally, we provide information on related work and conclusion.

II. Requirements

Before we design an execution model, we first list the requirements for the model.

The first requirement is "anonymity" which means that both software providers and data providers do not know the identity of consumers before/during/after the execution. Please note that it is not necessary (and not likely) that the identity of consumers is also hidden from the resource provider.

The second requirement is that the execution should be fully logged. For software/data providers, although they do not know who has used their software/data, they should know how their software/data is used, and receive a detailed report about all related executions, including execution time, execution resource utilization, and so on. The information could be used for auditing and billing purpose.

The third requirement is that the model must provide a clear execution interface for software and data providers to follow. As long as the software and data follow the interface, the resource provider can start the execution by combining the software and data correctly. Here we do not worry about the semantics of the software/data or whether or not the execution will generate any meaningful results.

The fourth requirement is that the system should have a strong access control mechanism. To prevent software/data from exposing to illegal consumers, it is strongly recommended to adopt a MAC (Mandatory Access Control) model [5], which we are using in our prototype system.

Other requirements include

- Authentication. It is mandatory to identify the user before she can manage her own digital assets.
- Repository (software/data) management. A friendly user interface should be provided which allows users to manage their assets. The management function should also include the access control management.
- Description of software/data. It is important to explain software/data clearly so that consumers know what they are about and how to use them.
- Search. An advanced search function allows users to find software or data which their need to complete the work.

III. High-Level Architecture

In this section, we present the general architecture of the system.

We consider an architecture with the following types of participants: *software provider*, *data provider*, *coordinator*, and *resource provider*. The software provider owns the software and the data provider owns the data. The coordinator provides interfaces for the data provider to find matched software and the software provider to find matched data. The resource provider hosts software and data in its storage, and provides computing resources to execute software on data.

Figure 1 demonstrates the relationship between participants.

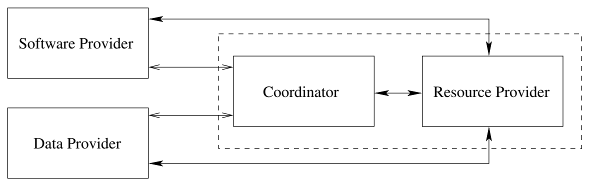

Figure 1. Four participants in the general architecture and the relationship among them.

In order to make software or data available for others to search and use, the software provider and the data provider must upload their software or data to the resource provider first. Here, both providers may decide to make their assets public, or keep them private. If the asset is private, it is only accessible by the owner (provider); if the asset is public, it is available to all authenticated consumers along with the owner. Please note that the policy about who can access the asset is defined by the owner at the upload time. Once software or data is uploaded, the resource provider will notify the coordinator about the newly added properties, and the coordinator starts to manage them.

There are two similar workflows. We describe how the data provider finds and executes software first. The data provider searches the coordinator and finds the software she needs. Please note that the coordinator applies certain access control rules so that the data provider can only see software that is public or shared with her. Once she finds the software, she can trigger the execution which applies the software on her own data. The data must be uploaded to the resource provider beforehand. The coordinator then returns an execution reference id to the data provider and she can use that id to track the execution status. Once the execution is complete, the output appears in the resource provider and by default, that output data is visible to the data provider only. She can view or download the output, and pay for the execution. The payment is split between the resource provider and the software provider. For the software provider, he knows that his software is used for a certain period of time, but he does not know who uses the software on which data set.

The flow about how the software provider finds data and apply software on the data is very similar and we will not repeat here.

In many cases, the coordinator is operated by the resource provider, but this requirement is not mandatory.

IV. Implementation Approach

Before we introduce our prototype implementation, we will first give a real-life use case that demonstrates the need for such an execution service. The Company F developed a tool to check bugs in web programs using advanced symbolic execution method. This tool requires a lot of computing and storage resources to build models for web programs, and the best place to host the tool is in cloud. A cloud-based version of the tool allows users to upload their web programs. After execution, a report with a list of potential bugs is generated and ready for users to download. For the web program developers, the tool is attractive, but using this tool in cloud means to give the source code of their web programs to an unknown party (the owner of the tool, company F). If the programs do contain bugs, it is difficult to prevent malicious attackers from using the bugs to attack the web system. For certain web sites, even letting outsiders know that the system has security related bugs is considered as a damage to the company's name. Therefore, currently this tool is limited for web programs are not very sensitive, or for the programmers who fully trust that company F will not abuse the tool and generated reports.

Our prototype system was developed using the Apache Struts framework, and is an example of a coordinator not operated by the resource provider. The front end offers a website like interface for data providers and software providers to upload their properties. The back end is an SQL database which manages the programs and data, as well as the registration and login information for the providers. The coordinator runs on an instance in Amazon's Elastic Cloud Compute (EC2).

The coordinator interface uses Apache Struts, an architecture to provide a standard MVC framework for use with JavaServer Pages (JSPs). MVC stands for Model-View-Controller framework, using JSPs and servlets together in order to deploy web applications. The Model portion of the system covers the internal state of the system, and actions which are available to change the state. The View is the user interface, and is built using JavaServer Pages. Apache Struts comprises the Controller portion of the system, dealing with user input.

The Model component is written in Java and controls the state and actions of the system. The internal state of the system is maintained in an SQL database; this is where all user information is stored, as well as all programs and data. In a coordinator operated by a cloud resource provider, one can imagine that the SQL database would be replaced with the cloud storage. The View component is comprised of JavaServer Pages. This is what the user sees when interacting with the coordinator. The Controller component is handled by Apache Struts, mapping user inputs to actions in the Model using ActionMappings. The Controller consults the ActionMappings when routing HTTP requests from one component to another in the framework. An ActionMapping consists of, at a minimum, a request path and an Action subclass to act upon the request. The request path tells the Controller which JSP page to redirect to, while the Action subclass indicates which class in the Model should execute.

A user, once registered, can upload data, programs, or both and mark them as shared or private. A private data or program is only visible to the owner. When a data or program is marked as shared, the user has an option to make it shared globally, which means that every user in the system can see and use it; or shared with a set of users (by giving the user names).

Once data and program are uploaded, they are encrypted by the coordinator before being stored. The keys are stored in a module called "accountability vault." There are two reasons for doing this step: first, we worry about the risks that are caused by insiders of resource providers bypass regular procedure and gain unauthorized access on software or data. It is quite difficult and expensive to fully monitor all nodes. By using encryption on the fly, we only need to protect the accountability vault which manages keys. This task is relatively easy. Second, all software or data access much go through the accountability vault to retrieve keys, which makes it a good place to log all events related to software and data.

Each program or data has a set of keywords and a category (such as Networking, Finance, etc) associated with it for searching purpose. Optionally, a data or program has an expiration option. Currently, we support two types of expiration: expired in a given time, or expired after a given number of execution reaches. When the expiration conditions are met, the program or data is marked as "unavailable" and is no longer visible to anyone other than the owner. The owner may choose to reactivate it (by changing the expiration condition), or do nothing, which will cause the data or program to be purged from the system after a period of time. Other properties include: detailed description, rule of charging users, and so on. The properties of data or programs can be modified by the owner.

An execution is triggered when a user selects a public program to apply on her private data, or she selects a public data to run her private program, or select a public program to run on a public data, giving the condition that she is authorized to use the public data and/or program. Whoever triggers the execution will be charged based on the predefined rules. If a user is using a public program on a public data, she will be charged for both assets. Before execution, a user has options to decide how to use the computing resources. If a user wants the result to be generated quickly, the execution will happen on an expensive VM setting with more CPUs and memories. And the execution time slot is guaranteed. Or if a user does not want to pay much, the execution will happen on a regular VM and start when the overall system load is low. For different options, the estimated execution time, which is based on previous statistics, and total cost are given.

A user can check the status of all executions she launched. The status includes: *pending*, *processing*, and *finished*. Pending means that an execution is planned, but not yet started; processing means that an execution has started but not finished; finished means that the execution is completed. A user can cancel all pending executions without being charged. Once an execution is finished, optionally, a user may choose to receive a notification through email. Please note that we do not distinguish a successful execution from a problematic execution because we expect that to be reflected in output.

When an execution is finished, the output by default is marked as private and the owner is the user who launched the execution. However, the software/data provider has check an option to see the output too. This requirement is from the real-life use case when a software provider wants to further tune the parameters based on output, and the output is not as sensitive as the original data.

For each finished execution, the resource utilization report is generated and shared with the program/data owner and the user who triggers the execution.

An important feature is the logging sub-system. Almost all events that are related to a program or data is logged, which include: generation, modification on properties, execution, expiration condition triggered, purged from the system, and so on. The log entries about execution only include the start/end time and the resource utilization, to protect the privacy of the other participant.

One feature we implemented is to allow software/data provider to choose how to purge expired software/data: use DoD 5220.22-M [6] method or simple file deletion. The is to protect certain providers' asset in an extreme way with extra cost attached.

For the use case we described in the beginning of the section, after we developed the prototype, we used the execution service to deploy the software validation tool in cloud.

V. Trusted Binding

In this section, we will discuss some ideas we have to further elevate the level of trust for resource providers.

The cloud paradigm is based on outsourcing computation and data storage. From a trust perspective the user loses control on where the data resides, when the data is processed, and what is the underlying trusted computing base (the set of hardware and software necessary to create a secure system). The cloud provider can create many instances of the algorithm (e.g. by replicating and launching virtual machines) and the user will never be able to notice the difference. The new virtual machine could be reading the data indistinctively in USA, China or India, and only the SLA could provide a relief based on trust in the cloud provider. Algorithms which are only intended to a specific data set due to intellectual property may be utilized indiscriminately by the cloud provider. An example is an algorithm to decode a protein from DNA pathway to create medication. A user could buy the algorithm instead of the medication itself to create medicines bound to a specific DNA. However, the cloud provider may replicate this algorithm in other data samples.

This situation is made more complex by mixing data to be executed and the actual algorithms or software that is going to execute the data. Even if the data is encrypted at rest, the algorithm needs to be provided with encryption key, or a handle to the encryption key. While at the moment of execution this access could be managed (by logging key usage by a specific algorithm) this enforcement could not be provided at execution time. After this moment the machine could be replicated in other hardware and the key will still exist in memory.

In the previous sections we proposed an architecture to separate algorithms and data, relying on the cloud provider (resource provider) to perform this separation, by implementing the relevant access control methods (utilizing a database, key repository, mandatory access control and encryption). With this necessary foundation we go one step further: enforcing data and algorithm matching at execution time, by only relying on a set of verifiable hardware mechanism and a minimal trusted computing base in the cloud.

The concept is as follows: every time an algorithm is launched, this algorithm is modified (imprinted) in a way which makes it able to read a data set, and only that data set. When the algorithm is executed, it is measured along with the underlying trusted computing base using chained measurements.

A. Trusted Hardware

State of the art trust assurance based on hardware-assisted mechanism has been around for a long time, in the form of the Trusted Platform Module [7]. The main concept in the trusted computing model is a root of trust, which initiates a set of chained measurements of the trusted computing stack. In clients, the Trusted Platform Module chip has been installed in most existing laptops, providing the possibility of creating these measurements.

However, the utilization ratio of TPM chips on laptop is very low, as the crypto module is off by default, and opting in is a lengthy process. Moreover, current clients execute millions of application, and this diversification makes difficult to reliably measure and verify the trusted computing base.

Trusted hardware can be available for the server for a very specific purpose: To obtain a cryptographically signed proof from the hardware that the trusted computing stack has not been modified. The trusted computing stack contains the hardware, hypervisor, management kernel and virtual machines.

The trusted computing module has an additional capability called sealing [8]. When sealing a data set, the data is cryptographically bound to a certain measurements or platform configuration. The data is only released if this platform is properly configured.

The use of trusted hardware is a leap forward in assuring the integrity of data at the moment of execution, but it falls short in ensuring access control for data. In the moment of measuring, a naive approach could consist of opening the data into the software, and measuring the virtual machine in a whole. Another approach is just measuring the software, and then executing the data. Both approaches fail to provide a one to one binding to ensure data privacy, from both the software provider and the resource provider. It will ensure however the integrity of the software, but not the wrong utilization of the software (using the software in other data sets)

B. Trusted binding of data and software

Software can be given access to several data sets. We suppose that this data sets are encrypted and have a unique hash value, i.e., are not modified. In the moment the software is given access to a new data set, the software is modified uniquely for this data set. The software should perform the same functionality, and no performance reduction by this modifications. Several techniques exist to provide this functionality, such as packing and scrambling. This can be done at the API level for the software or at the virtual machine level or as a combination of the two. The owner will be notified of this new software instance and the measurements for the instance. Notice that as some software may have policy defining the number of times they can be executed on a certain data set, or that they could be executed only during a time period against a specific data set, this provide a reliable logging mechanism. If the period expires, the modified instance could just be erased. Other instances will not be able to process that data set. Mechanisms to ensure data exist or is erased is currently subject to vigorous research [9].

Every time this specific software instance is called, the measurements will readily reflect which data it is processing. Once this new instance is launched, the key for the data is sealed to this specific configuration in to the cryptographic chip. Hence, the key for the data set could only be retrieved if this specifically modified software is executed. Notice that only specific machines will be able to open the sealed data. That way the user will now exactly know how many machines and what configuration is executing a specific software reading a data set. This protocol may reduce the performance slightly, as data encryption key must be first sealed to a specific hardware before the data could be read. This sealing migration could be done utilizing previously proposed protocols [10].

VI. Related Works

A. Data privacy in SaaS

Differentiation of data and application as a solution for data privacy was proposed in [11]. However this work falls short on describing an architecture for sharing algorithms, and does not describe any methods for reliable data/software binding. The problem was tackled differently in [12], where the authors propose a multilayered encryption scheme to preserve privacy in multitenant SaaS offerings. This approach is applicable to large databases utilized by current SaaS approaches. Our design supposes static data uploaded by the users instead of dynamically changing databases, so only a dynamic binding from data and software is necessary.

B. Mandatory Access Control (MAC)

Mandatory access control (MAC) [5] refers to enforcing access control whenever a subject accesses or perform operations on an object. Compare with discretionary access control (DAC), which allows subjects to make policy decision and/or assign security attributes, MAC-enabled systems allow policy administrators to implement fine-grained security policies.

A simple MAC model is the access matrix model, first proposed by Lampson [13] and later refined by Graham and Denning [14]. The basic idea is to use a matrix to define the authorization state. Each row of the matrix represents a user and each column represents one object (program and/or data files). The content of the cell is the authorization a user has on an object. Although the model looks primitive, it gives a nice abstract of a protection system. In our prototype system, we adopted this model as the back-end policy determination engine.

For a large system, we may also consider using Role-Based Access Control (RBAC) to enforce mandatory access control policies, as proposed in [15].

C. Cloud computing and SaaS

Cloud computing refers to both the applications delivered as services over the Internet and the hardware and systems software in the datacenters that provide those services [16]. And the services are referred to as Software as a Service (SaaS) [17].

The business model of SaaS was discussed in [18]. [19] showed the difference between SaaS and traditional software, and examined short- and long-term competition between them. Our paper suggested an alternative model to deliver software through a trusted resource provider.

VII. Conclusion

Currently, the existing mechanisms to raise the trust level for SaaS rely on applying a voluntarily self-regulation or passing a third-party auditing process, which is very costly. In this paper, we introduced a new mechanism to utilize software/data. The platform allows consumers to use software and/or data in cloud without the direct involvement of providers. This also allows to introduce trusted hardware to allow data/software unique bindings and trusted certification for execution. To demonstrate the idea, we developed a prototype system built and deployed in Amazon EC2 along with algorithms that are sensitive in nature. In the future, we plan to incorporate the idea of trusted binding into the prototype.

References

[1] M. Turner, D. Budgen, and P. Brereton, "Turning Software into a Service," *Computer*, vol. 36, no. 10, pp. 38–44, 2003.

[2] S. Pearson, "Taking account of privacy when designing cloud computing services," in *Proceedings of 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*, 2009, pp. 44–52.

[3] D. Baker and A. Sali, "Protein structure prediction and structural genomics," *Science's STKE*, vol. 294, no. 5540, p. 93, 2001.

[4] O. Tkachuk and S. P. Rajan, "Application of automated environment generation to commercial software," in *Proceedings of the 2006 international symposium on Software testing and analysis*. ACM, 2006, p. 214.

[5] D. E. Denning, "A lattice model of secure information flow," *Commun. ACM*, vol. 19, no. 5, pp. 236–243, 1976.

[6] Online, "Dod 5220.22-m national industrial security program operating manual (nispom)," 1995, http://www.usaid.gov/policy/ads/500/d522022m.pdf.

[7] Trusted Computing Group, "Trusted platform module (TPM) specifications," Online, *https://www. trustedcomputinggroup.org/specs/TPM/.*

[8] D. Challener, K. Yoder, R. Catherman, D. Safford, and L. Van Doorn, *A practical guide to trusted computing*. IBM Press, 2007.

[9] K. D. Bowers, A. Juels, and A. Oprea, "Proofs of retrievability: theory and implementation," in *CCSW '09: Proceedings of the 2009 ACM workshop on Cloud computing security*. New York, NY, USA: ACM, 2009, pp. 43–54.

[10] U. Kuhn, K. Kursawe, S. Lucks, A. Sadeghi, and C. Stuble, *Cryptographic Hardware and Embedded Systems*. Springer Berlin, 2005, vol. 3659/2005, ch. Secure Data Management in Trusted Computing.

[11] Z. Qiang and C. Dong, "Enhance the user data privacy for saas by separation of data," in *Information Management, Innovation Management and Industrial Engineering, 2009 International Conference on*, vol. 3, 26-27 2009, pp. 130 – 132.

[12] K. Zhang, Y. Shi, Q. Li, and J. Bian, "Data privacy preserving mechanism based on tenant customization for saas," in *Multimedia Information Networking and Security, 2009. MINES '09. International Conference on*, vol. 1, 18-20 2009, pp. 599 –603.

[13] B. Lampson, "Protection," in *5th Princeton Symposium on Information Science and Systems*, 1974, pp. 437–443.

[14] G. Graham and P. Denning, "Protection – principles and practice," in *Proc. Spring Jt. Computer Conference*, 1972, pp. 417–429.

[15] S. Osborn, R. Sandhu, and Q. Munawer, "Configuring role-based access control to enforce mandatory and discretionary access control policies," *ACM Trans. Inf. Syst. Secur.*, vol. 3, no. 2, pp. 85–106, 2000.

[16] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the clouds: A berkeley view of cloud computing," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28, Feb 2009. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html

[17] N. Gold, C. Knight, A. Mohan, and M. Munro, "Understanding service-oriented software," *IEEE Softw.*, vol. 21, no. 2, pp. 71–77, 2004.

[18] D. Ma, "The business model of software-as-a-service," in *2007 IEEE International Conference on Service Computing (SCC 2007)*, July 2007, pp. 701–702.

[19] M. Fan, S. Kumar, and A. B. Whinston, "Short-term and long-term competition between providers of shrink-wrap software and software as a service," *European Journal of Operational Research*, vol. 196, no. 2, pp. 661 – 671, 2009.