2010 Fifth IEEE International Symposium on Service Oriented System Engineering

Construction of a Trusted SaaS Platform

Chaoliang Zhong*, Jun Zhang†, Yingju Xia‡, Hao Yu§
Information Technology Laboratory
Fujitsu Research and Development Center
Beijing, China
{*clzhong, †jzhang, ‡yjxia, §yu}@cn.fujitsu.com

***Abstract*—As a part of Cloud computing, *Software as a Service*(SaaS) enables users to cut their costs by outsourcing software as a service on-demand, in which both computation and storage are handled on server-side. However, data must be decrypted into memory when performing the computation, even though they can be encrypted during storage and transmission. In this case, the privileged administrators of SaaS providers are able to inspect or modify users' data and computations. As a result, the users will not trust the SaaS providers, which is the number one factor blocking the wide adoption of SaaS.**

To address this problem this paper proposes a *trusted SaaS platform*(TSP). Besides guaranteeing data security during storage and transmission, TSP enforces a *trusted execution environment*(TEE) that guarantees the confidentiality and integrity of the users' data and computations.

***Keywords*-Cloud Computing; SaaS; Trusted Computing;**

I. Introduction

In recent software industry, *Software as a Service*(SaaS) model has been growing rapidly and seems to have a promising future. Rather than selling a software licence and installing the software on users' machines, the service provider operates an application on its infrastructure stack and licenses it to the users as a service on demand over Internet [1].

Although the users are attracted by the reduction of *Total Cost of Ownership*(TCO) [2] and other advantages of the adoption of SaaS(e.g., better resources utilization, more application access scalability, global outsourcing possibility, etc.) [3], the users are afraid of the loss of control of their data after adopting SaaS services, because SaaS applications process users' data on machines that are owned by a SaaS provider and may be physically thousands of miles away from the users [4]. That is to say, the users are still hesitating to put their data into the SaaS cloud. The users' concerns are reasonable, because their data, especially business sensitive data are the core competitive power for them. It could damage the users' companies if the business sensitive data are leaked when the users are using a SaaS application. Here, the SaaS users' demand is simple but challenging: nobody(even the administrators of SaaS providers) but the owner of the data knows the content of the data.

There is an analogy that is widely used to persuade the users to trust the SaaS providers: your data are more secure in the cloud, just like your money is more secure in the bank [5]. This kind of persuasion makes no sense, because the money a user deposits in his bank account is not different from any others' money, while a user's data are unique, confidential, business sensitive and different from any other's data essentially. Therefore, although top SaaS providers are continuously constructing their reputation to build the users' confidence(e.g., Google Docs deals with security issues [6]) and *service level agreement*(SLA) is now widely used to govern the SaaS providers' use of users' data at the legal level [4], nothing but a technical solution that guarantees the confidentiality and integrity of data in SaaS model can convince the hesitating users to adopt SaaS model.

Data encryption may be an effective approach to protecting data security for SaaS users during transmission and storage [7], [8]. Some research has also been done on computing on encrypted data [9], though currently there is not a mature solution to this issue. Moreover, after all, the unencrypted data are easier to process for SaaS providers. Given that SaaS providers are more willing to provide the services that can only process the users' unencrypted data, and the data must reside in the memory of the SaaS provider's host during computation, and anyone(e.g., administrators of the host) who can access this host can inspect or modify the users' unencrypted data, this paper attempts to provide a solution that guarantees the confidentiality and integrity of users' data during computation in SaaS model.

Conventional trusted computing platforms like *Terra* [10] are able to prevent the owner of a physical machine from inspecting or interfering with a computation running in a virtual machine(VM) that is hosted in the physical machine, and thus can effectively secure the computation running in the VM. However, these platforms cannot address this problem due to the following two reasons. First, they do not specify who will launch the VM that is responsible for performing the computation. If the VM is launched by a SaaS provider, as the owner of the VM, the SaaS provider can certainly inspect or modify the computation. Second, in a conventional SaaS system, any application needs a model of computation, a model of storage and a model of communication [11], [12], just as the simplified architecture shown in Fig. 1, in which a conventional SaaS system consists of three kinds of servers, storage, compuataion

978-0-7695-4081-8/10 $26.00 © 2010 IEEE
DOI 10.1109/SOSE.2010.34

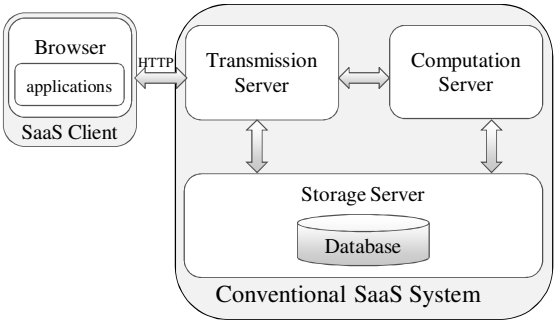

Figure 1. Conventional SaaS System

and transmission respectively. Although these platforms can guarantee each of the three servers is trusted, they cannot guarantee the SaaS system is trusted as well. For example, the transmission server is able to transfer the user's data to an untrusted computation server, even though itself is trusted. Actually, transmission and storage servers are not neccessary to be trusted.

The approach proposed in *Towards Trusted Cloud Computing* [13] can only be used for IaaS and cannot solve this problem due to the following reasons. The first reason is the same as that of conventional trusted computing platforms, that is to say, TCCP does not specify who will launch the VM either. Second, the architecture used in TCCP is Client/Server model and it is easy to implement data encryption with user's private trusted key on client-side. In SaaS system, however, the client is a browser, in which the encryption is only used for transmission(e.g., TLS [14]). Third, the protocols presented in TCCP cannot be used to solve this problem. In TCCP, the protocols are mainly utilized for node registration and securing VM launch and migration. However, in SaaS system, the users' main purpose is guaranteeing that the SaaS providers process their data and respond the result without inspection or modification, rather than guaranteeing the security of their VMs.

To address this problem, this paper proposes a *trusted SaaS platform*(TSP) that enables a trusted third party to launch a VM as a *trusted execution environment*(TEE) on the computation server. Though the privileged administrators of SaaS providers can access the physical host of TEE, they cannot access the TEE because the TEE is not launched by them. By taking advantage of trusted virtual machine monitor(TVMM) [10](see section II-B), the privileged administrators cannot tamper with the TEE. The TEE is also where all of the decryption, computation and encryption take place, so it can ensure the confidentiality and integrity of users' data and computations that are outsourced to SaaS services. Moreover, the TSP guarantees the data stored on the storage server and the data exchanged among these entities(i.e., SaaS client, transmission server, storage server and computation server, etc.) are all encrypted by taking advantage of a USB device on the SaaS client and protocols among these entities, so that the data won't be inspected or modified during transmission and storage.

In this paper we show how to design the TSP by taking advantage of trusted computing technologies. Section II introduces SaaS and one of the open source implementations of IaaS called Eucalyptus, introduces the related technologies, and presents the essential conflict between SaaS users and providers. Section III presents our design of TSP. Finally, section IV concludes this paper and discusses about the future work.

II. Background

A. Cloud Computing

Cloud computing refers to both the applications delivered as services over the Internet and the hardware and system software in the datacenters that provide those services [11]. In the higher layer of cloud computing, the services themselves have long been referred to as SaaS, while in the lower layer, the services that deliver the hardware to users and allow users to have access to entire VMs hosted by the providers are reffered to as Infrastructure as a Service(IaaS).

Data confidentiality and auditability is one of the top 10 obstacles to the adoption of cloud computing [11]. Nuno Santos et al. [13] has proposed a solution for IaaS, while there is no ideal solution proposed for SaaS currently.

1) Software as a Service: In the higher layer of cloud computing, SaaS(e.g., salesforce.com, etc.) offers users complete online applications as services over Internet [1]. Both the users and providers can benefit from moving to the SaaS model(e.g., the reduction of TCO [2], [3]). However, the users are still hesitating to put their data into the SaaS cloud, since they are afraid of the loss of control of their data after adopting SaaS services [4].

In a SaaS system, any application needs a model of computation, a model of storage and a model of communication [11], [12]. According to the three models, Fig. 1 presents a simplified architecuture of conventional SaaS systems, which includes three kinds of servers, namely storage, compuataion and transmission, respectively. Our solution to this problem is based on this simplified architecture, in which the SaaS client is the users' terminal and contains a WEB browser. The applications providing users with a *graphical user interface*(GUI) is running on that browser. The WEB browser communicates with the transmission server to send requests and receive responses. The SaaS system handles users' requests and responds the results with the collaboration of the three kinds of servers.

From the perspective of users, the SaaS system provides three kinds of operations, namely *data storage*, *retrieval* and *process*. The solution proposed in this paper also aims to guarantee the data security when performing these three operations.

2) Infrastructure as a Service: In the lower layer of cloud computing, IaaS providers such as Amazon allow their users to have access to entire VMs hosted by the providers. A user of the system is responsible for providing the entire software stack running inside a VM.

IaaS is also faced with the similar problem of SaaS. Fortunately, a solution to this problem for IaaS has already been proposed in [13]. Thus, in this paper, we do not focus on IaaS. Instead, we leverage one of the open source implementations of IaaS, namely Eucalyptus [15], to launch the VM of TEE in our solution.

The system of Eucalyptus manages one or more clusters whose nodes run a virtual machine monitor(VMM) to host VMs. Eucalyptus comprehends a set of components to manage the clusters. A simplified architecture of Eucalyptus is presented in [13], in which all those components are aggregated in a single *cloud manager*(CM) that handles a single cluster. There is a *node controller*(NC) that communicates with CM to execute the commands of CM such as launch, terminate VMs on each node of the cluster. Readers can refer to [15] for more details.

B. Trusted Computing

Trusted computing developed and promoted by Trusted Computing Group(TCG) [16] involves a set of hardware and software technologies that enable the construction of trusted platforms that can guarantee computers consistently behave in expected ways. In particular, the TCG proposed a standard for the design of *trusted platform module*(TPM) chip that is now bundled with commodity hardware. The TPM contains an endorsement private key(EK) that uniquely identifies the TPM(thus, the physical host), and some unmodifiable cryptographic functions. The respective manufacturers sign the corresponding public key to guarantee the validity of the key and correctness of the chip.

By leveraging the TPM chips, trusted platforms [10], [17] allow changes to the computers to be detected by authorized parties. This mechanism called *remote attestation* works as follows. At boot time, the host computes a measurement list ML containing a sequence of hashes of the software involved in the boot sequence, namely the BIOS, the bootloader, and the software implementing the platform. The ML is securely stored inside the host's TPM. To attest to the platform, a remote party challenges the platform running at the host with a nonce n_U. The platform asks the local TPM to create message containing both the ML and the n_U, encrypted with the TPM's private EK. The host sends the message back to the remote party who can decrypt it using the EK's corresponding public key, thereby authenticating the host. By checking that the nonces match and the ML corresponds to a configuration it deems trusted, a remote party can reliably identify the platform on an untrusted host.

A trusted platform like Terra [10] implements a *trusted virtual machine monitor*(TVMM) that enforces a *closed box* execution environment, meaning that a guest VM running on top cannot be inspected or modified by administrators with full privileges over the host. The TVMM guarantees its own integrity until the machine reboots. Thus, remote parties can attest to the platform running at the host to verify that a TVMM is running, and thus make sure that their computation running in a guest VM is secure.

Given that a conventional trusted platform can secure the computation in a VM on a single host, a intuitive approach to securing the computations of SaaS services is to deploy the platform at each computation server inside a SaaS system and perform the computation in the secured VM. However, this approach is insufficient. First, the party who ought to launch the VM is not specified and the party obviously cannot be the SaaS provider, because if the VM is launched by the SaaS provider, as the owner of the VM, the SaaS provider can certainly inspect or modify the computation. Second, there is a clear need for a mechanism that enforces the computation to be performed in the VM rather than outside it. Therefore, the TSP needs to provide a sequence of protocols that can satisfy the above two requirements.

C. Trusted Cloud Computing Platform

The *trusted cloud computing platform*(TCCP) [13] provides a closed box execution environment by extending the concept of trusted platform to an entire IaaS backend. The TCCP guarantees the confidentiality and integrity of a user's VM, and allows a user to determine up front whether or not the IaaS enforces these properties.

The trusted computing base of the TCCP includes two components: a TVMM and a *trusted coordinator*(TC). Each node of the backend runs a TVMM that hosts users' VMs, and prevents privileged users from inspecting or modifying them. The TVMM protects its own integrity over time, and complies with the TCCP protocols. That is to say, the TVMM is not only a VMM that hosts the VMs but also a component with the functionality of NC that communicates with the CM to receive and execute the commands such as launch, terminate VM.

The TC manages the set of *trusted nodes* that can run a user's VM securely. To be trusted, a node must be located within the security perimeter, and run the TVMM. To meet these conditions, the TC maintains a record of the nodes located in the security perimeter, and attests to the node's platform to verify that the node is running a trusted TVMM implementation. A user can verify whether the IaaS service secures its computation by attesting to the TC.

To secure the VMs, each TVMM running at each node cooperates with the TC in order to 1) confine the execution of a VM to a trusted node, and to 2) protect the VM state against inspection or modification when it is in transit on the network. The critical moments that require such protections are the operations to *launch*, and *migrate* VMs. The TCCP specifies several protocols to secure these operations.

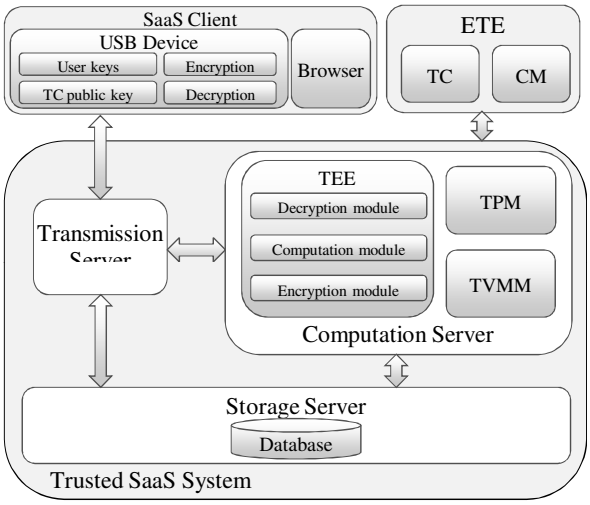

Figure 2. Trusted SaaS Platform

TCCP assumes an *external trusted entity*(ETE) that hosts the TC, and securely updates the information provided to the TC about the set of nodes deployed within the IaaS perimeter, and the set of trusted configurations. TCCP envisions that the ETE should be maintained by a third party with little or no incentive to collude with the IaaS provider. Thus, sysadmins that manage the IaaS have no privileges inside the ETE, and therefore cannot tamper with the TC.

The TSP takes advantage of the trusted computing base of the TCCP and the computation server that hosts the TEE is treated as a trusted node. However, the protocols presented in TCCP are not suitable for TSP, because they are mainly utilized for node registration and securing VM launch and migration, while the main purpose of TSP is to guarantee that the SaaS providers process users' data and respond the result without inspection or modification, namely the security of the three operations(i.e., data storage, data retrieval and data process) provided by SaaS providers. Therefore, the TSP needs to present new protocols that are suitable for this purpose.

D. Essential conflict and our solution

The administrators of SaaS providers have privileged control over all of the transmission, storage and computation servers. The SaaS providers require the data provided by users are not encrypted or can be decrypted by them. In this case, the privileged administrators even need not to install special software to perform an attack and thereby access SaaS users' data or modifiy the computation. As a result, the SaaS users are not willing to provide the unencrypted data or the data that can be decrypted by SaaS providers.

This is the conflict between the confidentiality and integrity of data required by the SaaS users and the decryptable and computable data required by the SaaS providers. Assuming that the data have been encrypted with a public key of a trusted third party besides the transmission encryption(e.g., TLS [14]) on SaaS client, in order to solve this conflict, our solution is to seperate the running control of the computation from the design of the computation. The design of the computation means how to process the unencrypted data and certainly should be completed by the SaaS providers, while the running control of the computation means who and how to control the running of the computation and should be accomplished by the trusted third party rather than the SaaS providers. The unencrypted data are required only when the computation is running and the running of the computation is controlled or protected by a trusted third party, so that the SaaS providers are not able to inspect or modify the data and computation. The TSP presented in section III provides a mechanism that implements this seperation.

III. Trusted SaaS Platform

We present the TSP that guarantees the confidentiality and integrity of a user's data and computations by enforcing a TEE which is a VM launched by a trusted third party on the computation server. The TSP enhances the simplified architecture of the conventional SaaS system to enable the TEE without greatly changing it. The TSP also guarantees the data stored on the storage server or exchanged among entities of a SaaS system are all encrypted, so that the data cannot be inspected or modified during transmission and storage. From the perspective of users, a SaaS system provides three kinds of operations, namely *data storage*, *retrieval* and *process*. These operations are accomplished with the collaboration of the entities of the SaaS system. The TSP presents several protocols for guaranteeing the security of these operations by specifying the communications among the entities.

A. Overview

As depicted in Fig. 2, the TSP includes the entities that are also involved in conventional SaaS systems: *SaaS client*, *SaaS transmission server*(STS), *SaaS storage server*(SSS) and *SaaS computation server*(SCS). By drawing lessons from TCCP [13], the TSP also involves a new entity called *external trusted entity*(ETE), which hosts a remote *trusted coordinator*(TC) and a *cloud manager*(CM).

The TSP takes advantage of trusted computing to enforce the TEE on SCS. The trusted computing base of the TSP includes two components: a TVMM and a TC. The TVMM runs on the SCS, hosts the VM providing the TEE and prevents privileged users of the physical host of SCS such as the administrators of SaaS providers from inspecting or modifying the memory of TEE. The TVMM protects its own integrity over time. The SCS embed a certified TPM chip and must go through a secure boot process to install the TVMM. Due to space limitations we will not go into detail about the design of the TVMM, but readers can refer to [17] for an architecture that can be leveraged to build a TVMM.

The TC manages the SCS that can be trusted and run the VM of TEE securely. In order to be trusted, the SCS must run a TVMM and register with the TC when its physical machine is being launched. After that TC will attest to the TPM of SCS to verify that the SCS is running a TVMM. The related protocol is presented in section III-C1.

Besides hosting and protecting the VM of TEE, the TVMM can receive and execute the commands(e.g., launch, terminate VM, etc.) of CM as well. In the final step of SCS registration, if the SCS is reckoned to be trusted, TC will luanch the VM of TEE on the SCS by asking the CM to send a command to the TVMM. In that command, the information for launching the VM is included. The information involves an image file that is stored on the CM before being transferred to SCS and contains three modules, namely decryption, computation and encryption. In TSP, the computation module is required to be provided by the SaaS provider. Therefore,

- Since the VM of TEE is launched by the TC, the administrators of the SaaS provider are not able to be logged in it.
- Moreover, because the TVMM is running, the administrators of the SaaS provider are not able to inspect or modify the data and computations in the VM of TEE.
- However, because the computation module is provided by the SaaS provider, the VM can execute the computation of the SaaS provider to process the users' data.

In the above mechanism, the running of the computation is controlled by the ETE, while the design of the computation is completed by the SaaS provider. Consequently, the running control of the computation is seperated from the design of the computation, and the confidentiality and integrity of users' data and computations are guaranteed.

B. Entities

The entities involved in TSP are controlled by different parties respectively. The SaaS client is controlled by the SaaS users. The ETE is controlled by the trusted third party. The trusted SaaS system which consists of STS, SCS and SSS is controlled by the SaaS providers.

SaaS Client: As the terminal of SaaS users, the SaaS client contains two components: a WEB browser and a USB device. The SaaS applications that provide users with a *graphical user interface*(GUI) run on the WEB browser. The USB device is provided by the trusted third party. In the USB device, there are *user private-public keys* of an asymetric cryptographic keypair, a *TC public trusted key*, decryption and encryption functions and a URI of TC. The keys and functions stored in the USB device are utilized to encrypt the messages from SaaS client to STS or decrypt the messages from STS to SaaS client. In particular, the HTTP requests from SaaS client consist of the data and operations that are encrypted with a generated session key. The generated session key encrypted with the *TC public trusted key* is a symetric cryptographic key and is sent together within the encrypted HTTP requests to STS.

SaaS Transmission Server: The STS is a WEB server that receives the requests from SaaS client directly. On one hand, since the contents of these requests are encrypted and the STS is not able to decrypt them, the STS has no choice but to forward them to a trusted SCS that can handle these requests. On the other hand, the result of computation from SCS or result of storage from SSS are also encrypted, the STS has no choice but to forward them to the SaaS client. Thus, the above mechanism guarantees the confidentiality and integrity of requests and responses relayed by STS. Besides, since the STS can detect the real-time load of each SCS and SSS when the SaaS system is running and the number of SCS and SSS can be dynamicly changed on-demand, the SaaS system can benifit from load balancing and scalability.

SaaS Computation Server: The SCS is a server running the VM of TEE. As depicted in Fig. 2, the SCS involves a TVMM and a TPM. The functionality of TVMM includes:

- hosting the VM of TEE.
- guaranteeing the data and computations in the VM of TEE will not be inspected or modified.
- complying with the protocol of SCS registration(see section III-C1), receiving and executing the commands of CM(e.g., launch VM, terminate VM, etc.).

The functionality of TPM is *remote attestation*(see section II-B). In order to be trusted, the SCS must run a TVMM and register with TC when its physical host is being launched. TC then attests to the TPM to verify that the SCS is running a trusted TVMM(see section III-C1). In the final step of SCS registration, if the SCS is trusted, TC will launch the VM of TEE by asking the CM to send a command to the TVMM.

When receiving the requests that are sent from the SaaS client, forwarded by the STS and encrypted with a generated session key, the SCS invokes the TEE to handle the requests. The TEE decrypts the data with the decryption module, processes them with the computation module, encrypts the result of the computation module with the encryption module and returns the encrypted result to the SCS. Finally, The SCS responds the result to the STS or stores them on SSS.

In the TEE, only the encryption module and the decryption module that are provided by the trusted third party can interact with the SCS, while the computation module provided by the SaaS provider is a internal module and cannot interact with the outside of TEE. Thus, the unencrypted data in the TEE are secured and impossible to be leaked.

SaaS Storage Server: The SSS is a server that handles the retrieval and storage requests and operates the database servers directly. These requests either come from SCS directly or come from SaaS client indirectly. All data stored on SSS are encrypted with users' public trusted key and can only be decrypted with users' private trusted key on SaaS client, which ensures the security of data during transmission

Table I
NOTATIONS

Notation	Description
$\langle K^p, K^P \rangle$	Private-public keys(asymetric)
$\{y\}_{K^x}$	Data y are encrypted with key K^x
EK_x	Endorsement keys
TK_x	Trusted keys
K_x	Session key(symetric)
n_x	Unique numbers generated by x
$data$	User's unencrypted data
loc_of_data	Where the SSS can find the data
op	Operation:storage,retrieval,process,etc.
str_op	Storage operation
op_res	The result of operation
str_op_res	The result of storage operation
$comp\_res$	The result of TEE's computation module
ML_x	Measure list of x
U_{user}	User identifier provided by user himself
U_{STS}	User identifier provided by STS

and storage. The public data that need not to be encrypted may be separated from the users' private data and stored on another database server by extending this architecture.

External Trusted Entity: The ETE hosts two components: TC and CM. The TC can attest to the TPM of SCS to verify that the TVMM is running when the SCS is launched and sends a registration request to the TC. If the TVMM is running, the SCS is reckoned to be trusted and the TC will launch the VM of TEE on the SCS by asking the CM to send a command to the TVMM. The ETE should be maintained by a third party with little or no incentive to collude with the SaaS provider. The administrators that manage the SaaS system have no privileges inside the ETE, and therefore cannot tamper with the TC and CM.

C. Message Flows

In this section, We describe the protocol that TC uses to manage the SCS (SCS registration) and the protocols that secure the operations involving data storage, data retrieval and data process. When performing these operations, the TSP needs to guarantee that 1) the data are always encrypted in the SaaS system but outside the TEE. 2) the data can be decrypted in the TEE. To satisfy these requirements, the parties involved in these operations follow the protocols depicted in Fig. 4, 5 and 6. These protocols are designed on the fact that, among the components of the service, the user only trusts the TC. In these protocols, we use the notations described in Table I for cryptographic operations.

1) SCS registration: The TC dynamically manages the set of trusted SCSes that can host a VM of TEE by maintaining a directory containing the public endorsement key EK^P_{SCS} identifying the TPM of SCS, and the expected measurement list ML_{SCS} for each SCS. The ETE makes some properties of the TC securely available to the public, namely the EK^P_{TC}, the ML_{TC}, and the TK^P_{TC}(identifying the TC). Both the ML_{SCS} and the ML_{TC} express the canonical

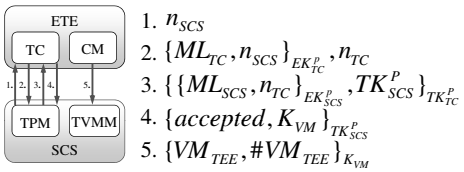

Figure 3. SCS Registration

configurations that a remote party is expected to observe when attesting to the platform running on a SCS or on the TC, respectively.

To be trusted, SCS must register with TC by complying with the protocol depicted in Fig. 3. In step 1 and 2, SCS attests to the TC to avoid an impersonation of the TC by an attacker: SCS sends a challenge n_{SCS} to the TC, and the TC replies with its bootstrap measurement ML_{TC} encrypted with EK^p_{TC} to guarantee the authenticity of the TC. If the ML_{TC} matches the expected configuration, it means the TC is trusted. Reversely, the TC also attests to SCS by piggybacking a challenge n_{TC} in message 2, and checking whether the SCS is authentic, and is running the expected configuration(step 3). The SCS generates a key pair $\langle TK^p_{SCS}, TK^P_{SCS} \rangle$, and sends its public key to the TC. If both peers mutually attest successfully, the TC adds TK^P_{SCS} to its SCS database, and sends message 4 containing a session key K_{VM} generated by the TC to confirm that the SCS is trusted. Key TK_{SCS} certifies that the SCS is trusted.

In the final step of SCS registration, if the SCS is trusted, the TC asks the CM to send message 5 to the TVMM of SCS. The message 5 contains VM_{TEE} and VM_{TEE}'s hash encrypted with the session key K_{VM} to guarantee its confidentiality and integrity during transmission. The TVMM then launches the VM of TEE with its initial state VM_{TEE} by asking TPM for K_{VM} to decrypt the message 5. The initial state VM_{TEE} contains a VM image file and a trusted keypair of TEE $\langle TK^p_{TEE}, TK^P_{TEE} \rangle$. A copy of the public trusted key of TEE TK^P_{TEE} is reserved in the TC's trusted TEE database corresponding to the trusted SCS that hosts the TEE, so that TC can authentic the identity of TEE when the TEE requires the TC to decrypt the session key as described in section III-C2 and III-C4.

In the case that a trusted SCS reboots, the TSP must guarantee that the SCS' configuration remains trusted, otherwise the SCS could compromise the security of the TSP. To ensure this, the SCS only keeps TK^p_{SCS} in memory causing the key to be lost once the machine reboots. The SCS is thus banned from the TSP, since it will not be able to decrypt messages encrypted with the previous key, and must repeat the registration protocol.

2) Data storage: In this section, we present the protocol depicted in Fig. 4 to secure the data storage operation.

First, the user's SaaS client generates a session key K, and sends message 1 to STS containing the following three

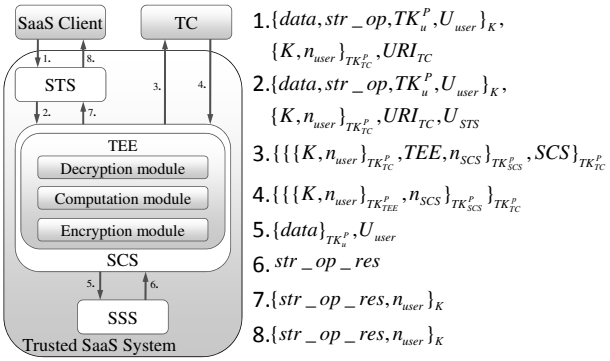

Figure 4. Data Storage

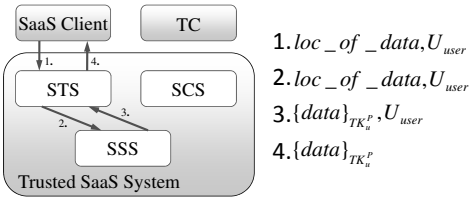

Figure 5. Data Retrieval

parts. The first part is encrypted with K and contains *data*, *operation*, *user's public trusted key* TK_u^P and user identifier U_{user} provided by user himself. The second part is encrypted with *TC's public trusted key* TK_{TC}^P and contains the session key K. The third part is the URI_{TC} without encryption. Encrypting the session key K with TK_{TC}^P ensures that only the TC can authorize someone to access the first part including the user's data and operation. The TC only authorizes trusted SCSes. The URI_{TC} indicates which TC should SCS communicate with for acquiring the K.

Upon receiving message 1 from SaaS client, since the STS cannot decrypt this message, it has no choice but to forward the message to a trusted SCS. Before forwarding, the STS adds an extra part U_{STS} to the message 1, which denotes the user's identity. Upon receiving message 2, since the SCS cannot decrypt this message outside TEE, it invokes the TEE to handle this message. In order to access the data and operation in the message, the decryption module of TEE sends message 3 to the TC that decrypts K on TEE and SCS's behalf. Message 3 is encrypted with TK_{SCS}^p, so that the TC can verify whether the SCS is trusted. If the corresponding public key is not found in TC's trusted SCS database, the request is denied. This would have been the case had the STS forwarded the message to an untrusted SCS. Otherwise, the SCS is reckoned to be trusted. Furthermore, the TC verifis whether the TEE exists and is running on the corresponding trusted SCS by searching in TC's trusted TEE database. If the TC is convinced that the TEE is running on the corresponding trusted SCS, it decrypts the K from message 3, encrypts the K with TK_{TEE}^P and TK_{SCS}^P, and sends message 4 containing the encrypted K to the TEE, such that only the TEE of the SCS can read the K. The decryption module of TEE is now able to decrypt and invoke the computation module to process the data, operation, TK_u^P and U_{user}.

Since the *operation* is a *storage operation* in this section, the computation module just checks whether the U_{user} equals to U_{STS} to prevent *masquerade attacks* [18] and then forwards the data, operation, TK_u^P and U_{user} to the encryption module directly if it is not an attack. The encryption module encrypts the data with TK_u^P to construct message 5 and sends it to SSS. The SSS stores the encrypted data associated with the user and responds the operation result. Finally, the TEE sends message 7 forwarded by STS to SaaS client containing the operation result to report whether the data storage operation is successfully executed.

3) Data retrieval: The protocol depicted in Fig. 5 is presented to secure the user's data retrieval operation. In practical systems, the data may be retrieved according to many features. In this paper, however, for description convenience, we assume that the user's data are always stored and retrieved according to their locations in database.

First, the user sends message 1 to STS containing the location of the requested data and the user's identifier. The STS checks whehter the identifier U_{user} is legal and its corresponding user has already been logged in the SaaS client to prevent *masquerade attacks* [18]. If it is not an attack, the STS forwards this message to SSS which queries the database to retrieve and return the encrypted data to STS in message 3. Upon receiving message 3, STS responds the encrypted data to the user's SaaS client. Finally, the user is able to access the requested data decrypted with the user's private trusted key stored in the USB device at SaaS client.

4) Data process: The protocol depicted in Fig. 6 is presented to secure the user's data process operation.

First, the user needs to fetch the data that need to be processed from the trusted SaaS system. The same as section III-C3, we assume that the user's data are always stored and retrieved according to their locations in database. From step 1 to 4, the user fetches the data with the protocol presented in section III-C3. After the data are fetched, the SaaS client decrypts the data with the user's private trusted key stored in the USB device, and sends message 5 to STS. The message 5 is similar to the message 1 of data storage protocol presented in section III-C2. The only difference between them is the *operation*. In data storage protocol, the *operation* is specified to *storage*, while in data process protocol, the *operation* is not specified. Instead, it may be any operation that the computation module can support.

The steps from 6 to 8 are the same as the steps from 2 to 4 of data storage protocol. After these steps are completed, the computation module of TEE can now access the decrypted data of message 5 and process them with the *operation*

1. loc_of_data, U_{user}

2. loc_of_data, U_{user}

3. $\{data\}_{TK_u^P}, U_{user}$

4. $\{data\}_{TK_u^P}$

5. $\{data, op, TK_u^P, U_{user}\}_K, \{K, n_{user}\}_{TK_{TC}^P}, URI_{TC}$

6. $\{data, op, TK_u^P, U_{user}\}_K, \{K, n_{user}\}_{TK_{TC}^P}, URI_{TC}, U_{STS}$

7. $\{\{\{K, n_{user}\}_{TK_{TC}^P}, TEE, n_{SCS}\}_{TK_{SCS}^P}, SCS\}_{TK_{TC}^P}$

8. $\{\{\{K, n_{user}\}_{TK_{TEE}^P}, n_{SCS}\}_{TK_{SCS}^P}\}_{TK_{TC}^P}$

9. $\{comp\_res\}_{TK_u^P}, U_{user}$

10. str_op_res

11. $\{op\_res, n_{user}\}_K$

12. $\{op\_res, n_{user}\}_K$

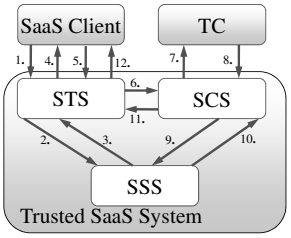

Figure 6. Data Process

that the message 5 requires. The result of computation module is encrypted by the encryption module with the user's public trusted key decrypted from message 5, and sent to SSS within message 9. The SSS stores this computation result according to the user identifier within message 9, and responds the operation result to SCS. Finally, the SCS returns the encrypted computation result to the SaaS client via STS with message 11 and 12.

IV. Conclusion and Future Work

In this paper, we argue that SaaS users' concerns about the confidentiality and integrity of their data and computations are the number one factor blocking the wide adoption of SaaS. We present the design of a *trusted SaaS platform*(TSP) that enforces SaaS providers to provide a *trusted execution environment*(TEE), in which the confidentiality and integrity of SaaS users' data and computation are guaranteed. We are implementing a fully functional prototype based on our design currently and plan to evaluate its performance in the near future.

References

[1] A. Konary, S. Graham, and L. Seymour, "The future of software licensing: Software licensing under siege," International Data Corporation, White Paper, Mar. 2004.

[2] Ellram and L. M, "Total cost of ownership: an analysis approach for purchasing," *International Journal of Physical Distribution and Logistics Management*, vol. 25, no. 8, pp. 4–23, 1995.

[3] D. C. Chou and A. Y. Chou, "Software as a Service (SaaS) as an outsourcing model: An economic analysis," in *Proc. SWDSI'08*, Houston, Texas, USA, Mar. 2008, pp. 386–391.

[4] B. R. Kandukuri, R. P. V., and A. Rakshit, "Cloud security issues," in *Proc. the 2009 IEEE International Conference on Services Computing*, Bangalore, India, Sep. 2009, pp. 517–520.

[5] T. Espiner. (2009, Nov.) Google: Data is more secure in the cloud. [Online]. Available: http://news.zdnet.co.uk/security/0,1000000189,39854667,00.htm

[6] J. Mann. (2009, Mar.) Google Docs deals with security issue. [Online]. Available: http://www.techspot.com/news/33839-google-docs-deals-with-security-issue.html

[7] F. Chong, G. Carraro, and R. Wolter. (2006, Jun.) Multi-tenant data architecture. [Online]. Available: http://msdn.microsoft.com/en-us/library/aa479086.aspx

[8] C. Wang, Q. Wang, K. Ren, and W. Lou, "Ensuring data storage security in cloud computing," in *Proc. IWQoS'09*, Charleston, South Carolina, USA, 2009.

[9] A. Sahai, "Computing on encrypted data," in *Proc. ICISS 2008*, Hyderabad, India, Dec. 2008, pp. 148–153.

[10] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh, "Terra: A virtual machine-based platform for trusted computing," in *Proc. SOSP'03*, 2003.

[11] 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, Technical Report UCB/EECS-2009-28, Feb. 2009. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html

[12] L. Youseff, M. Butrico, and D. D. Silva, "Toward a unified ontology of cloud computing," in *Proc. Grid Computing Environments Workshop on Supercomputing Conference(SC'08)*, Austin, Texas, USA, Nov. 2008, pp. 1–10.

[13] N. Santos, K. P. Gummadi, and R. Rodrigues, "Towards trusted cloud computing," in *Proc. HotCloud'09*, San Diego, CA, USA, Jun. 2009.

[14] T. Dierks and C. Allen, "The TLS protocol," Internet Engineering Task Force, RFC 2246, Jan. 1999. [Online]. Available: http://www.ietf.org/rfc/rfc2246.txt

[15] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "Eucalyptus: A technical report on an elastic utility computing architecture linking your programs to useful systems," UCSB Computer Science, Technical Report 2008-10, 2008.

[16] (2009) The TCG website. [Online]. Available: https://www.trustedcomputinggroup.org

[17] D. G. Murray, D. G. Murray, and S. Hand, "Improving xen security through disaggregation," in *Proc. VEE'08*, New York, NY, USA, 2008, pp. 151–160.

[18] E. Guttman, L. Leong, and G. Malkin, "Users' security handbook," Internet Engineering Task Force, RFC 2504, Feb. 1999. [Online]. Available: http://www.rfc-editor.org/rfc/rfc2504.txt