IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 17, Issue 6, Ver. II (Nov – Dec. 2015), PP 01-23
www.iosrjournals.org

Design & Development of a Trustworthy and Secure Billing System for Cloud Computing Architecture

Prof.Dr.G.Manoj Someswar[1], Firdous Rehana[2], Mohd.Abdul Kareem[3]
1. B.Tech., M.S.(USA), M.C.A., Ph.D., Professor & Dean (Research), Department Of CSE, Nawab Shah Alam Khan College of Engineering & Technology, Affiliated to JNTUH, Malakpet, Hyderabad, Telangana, India.
2. M.Tech. (CSE), Assistant Professor, Department Of CSE, Nawab Shah Alam Khan College of Engineering & Technology, Affiliated to JNTUH, Malakpet, Hyderabad, Telangana, India.
3. M.Tech. (CSE), Nawab Shah Alam Khan College of Engineering & Technology, Affiliated to JNTUH, Malakpet, Hyderabad, Telangana, India.

Abstract: *Cloud computing is an important transition that makes change in service oriented computing technology. Cloud service provider follows pay-as-you-go pricing approach which means consumer uses as many resources as he need and billed by the provider based on the resource consumed. CSP give a quality of service in the form of a service level agreement. For transparent billing, each billing transaction should be protected against forgery and false modifications. Although CSPs provide service billing records, they cannot provide trustworthiness. It is due to user or CSP can modify the billing records. In this case even a third party cannot confirm that the user's record is correct or CSPs record is correct. To overcome these limitations we introduced a secure billing system called THEMIS. For secure billing system THEMIS introduces a concept of cloud notary authority (CNA). CNA generates mutually verifiable binding information that can be used to resolve future disputes between user and CSP. This project will produce the secure billing through monitoring the service level agreement (SLA) by using the SMon module. CNA can get a service logs from SMon and stored it in a local repository for further reference. Even administrator of a cloud system cannot modify or falsify the data.*

Keywords: *Cloud Service Provider, Cloud Notary Authority, Service Level Agreement, Accounting Processor for Event Logs, Secure Virtual Machine Execution, GRASP, TISA, VIPM.*

I. Introduction

Cloud computing, or the cloud, is a colloquial expression used to describe a variety of different types of computing concepts that involve a large number of computers connected through a real-time communication network shows such as Internet.[1] Cloud computing is a term without a commonly accepted unequivocal scientific or technical definition. In science, cloud computing is a synonym for distributed computing over a network and means the ability to run a program on many connected computers at the same time. The phrase is also, more commonly used to refer to network-based services which appear to be provided by real server hardware, which in fact are served up by virtual hardware, simulated by software running on one or more real machines. Such virtual servers do not physically exist and can therefore be moved around and scaled up (or down) on the fly without affecting the end user - arguably, rather like a cloud. The popularity of the term can be attributed to its use in marketing to sell hosted services in the sense of application service provisioning that run client server software on a remote location.

Architecture Of Cloud Computing

It all starts with the front-end interface seen by individual users. The user's request then gets passed to the system management, which finds correct resources and then calls the system's appropriate provisioning services. These services slice out the necessary resources in the cloud, cast the appropriate web application and either creates or opens the requested document. The below figure shows the concept of cloud.

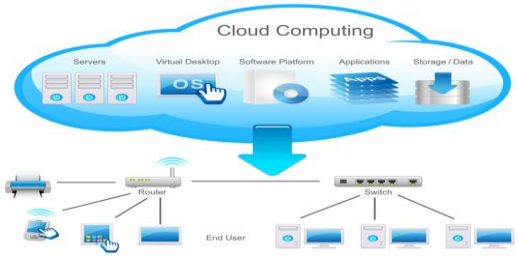

Figure 1: Architecture Of Cloud Computing

Service Models

Infrastructure-As-A-Service (IaaS)

THIS is the basic cloud service model and cloud providers offer computers, and as a physical or more often as virtual machines, and other resources. Other resources in IaaS clouds include images in a virtual machine image library, raw (block) and file-based storage, load balancers, IP addresses, virtual local area networks (VLANs), firewalls, and software bundles. Users use an IaaS may wish to figure 1.4 out the billed charges for the total service time and guaranteed service level. [1]

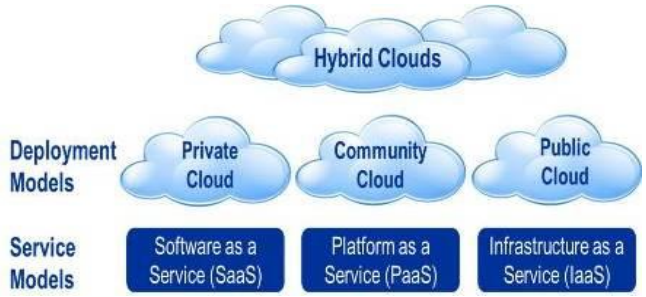

Figure 2: Service Model

PLATFORM AS A SERVICE (PAAS)

This is a category of cloud computing services that provide a computing platform and a solution stack as a service. Along with SaaS and IaaS, it's a service model of cloud computing. In this model, the consumer produces the software using tools and libraries from the provider. The consumer also charge software deployment and configuration settings. The provider contributes the networks, servers and archive.[2]

Software As A Service (SaaS)

Sometimes referred to as "on-demand software", it's a software delivery model in which software and associated data are centrally hosted on the cloud. SaaS is typically an acquired by users using a thin client via a web browser. SaaS is a common delivery model for many business applications, including auditing, collaboration, customer relationship management (CRM), management information systems (MIS), enterprise resource planning (ERP), invoicing, human resource management (HRM), content management (CM) and service desk management. If a company uses a PaaS or SaaS the accounting department of the company may require the service usage logs so as to verify the billed charges by clicking company's total number of running software program or platforms.[3]

Deployment Models

Private Cloud: It is entirely dedicated to the needs of a single organization. Private cloud can be on or off premises. An on-premise resides in the owner's computer room or data center and managed by the organizations own IT staff. An off-premise takes advantage of existing facilities and expertise of an outsourcing company such as co-location hosting facility. Advantage of private cloud is that an organization can design it, change it over time, control the quality of service provided. Disadvantages are it requires investment of expertise, money and duration.[4]

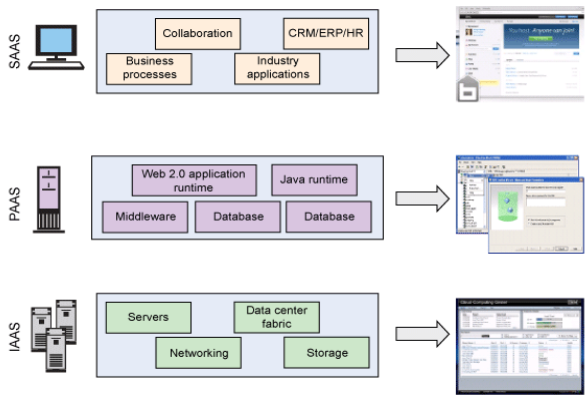

Figure 3: Deployment Diagram

Public Cloud

It is a multitenant cloud that is owned by a company that typically sells the services it provides to the general public. Advantage of public cloud is that it is always ready to use without delays, no need to invest in internal IT infrastructure.[5]

Hybrid Cloud

It is also known as cross-premises cloud uses a private and public cloud at the same time, with services spanning both deployments. Advantage of hybrid cloud are it control of security and compliance from private cloud, cost effective flexibility and scalability from public cloud and a single service spanning both.[6]

Community Cloud

It is owned, managed, shared and operated by many organizations. This open cloud can use many technologies. Community clouds are extremely complex and are a shared risk.[7]

Concept Of Cloud

Cloud Computing is the result of evolution and adoption of existing technologies and paradigms. The goal of cloud computing is to allow users to take benefit from all of these technologies, without the need for deep knowledge about or expertise with each one of them. The cloud aims to cut costs, and help the users focus on their core business instead of being impeded by IT obstacles.[8]

The main enabling technology for cloud computing is virtualization. Virtualization abstracts the physical infrastructure, which is the most rigid component, and makes it available as a soft component that is easy to use and manage. By doing so, virtualization provides the agility required to speed up IT operations, and reduces cost by increasing infrastructure utilization. On the other hand, autonomic computing automates the process through which the user can provision resources on-demand. By minimizing user involvement, automation speeds up the process and reduces the possibility of human errors.[26] Users face difficult business problems every day. Cloud computing adopts concepts from Service-oriented Architecture (SOA) that can help the user break these problems into services that can be integrated to provide a solution. Cloud computing provides all of its resources as services, and makes use of the well-established standards and best practices gained in the domain of SOA to allow global and easy access to cloud services in a standardized way. Cloud computing also leverages concepts from utility computing in order to provide metrics for the services used. [9] Such metrics are at the core of the public cloud pay-per-use models. In addition, measured services are an essential part of the feedback loops in autonomic computing, allowing services to scale on-demand and to perform automatic failure recovery.

Cloud computing is a kind of grid computing; it has evolved by addressing the QoS (quality of service) and reliability problems. Cloud computing provides the tools and technologies to build data/compute intensive parallel applications with much more affordable prices compared to traditional parallel computing techniques.[26]

Scope Of Research

Scope of the project is to provide a high securable and non obstructive billing system. Central Nodal Authority (CNA) generates the bill with binding information. The process, which involves a generation of mutually verifiable binding information among all the involved entities on the basis of a one-way hash chain, is computationally efficient for a thin client and the CSP. So even administrator of a cloud system cannot modify or falsify the data.

Vision

In this Technology for secure billing system THEMIS introduces a concept of cloud notary authority (CNA). CNA generates mutually verifiable binding information that can be used to resolve future disputes between user and CSP. This project will produce the secure billing through monitoring the service level agreement (SLA) by using the SMon module. CNA can get a service logs from SMon and stored it in a local repository for further reference.

Towards Trusted Cloud Computing

Cloud computing infrastructures enable companies to cut costs by outsourcing computations on-demand. However, clients of cloud computing services currently have no means of verifying the confidentiality and integrity of their data and computation. To address this problem we propose the design of a trusted cloud computing platform (TCCP). TCCP enables Infrastructure as a Service (IaaS) providers such as Amazon EC2 to provide a closed box execution environment that guarantees confidential execution of guest virtual machines. Moreover, it allows users to attest to the IaaS provider and determine whether or not the service is secure before they launch their virtual machine. A trusted cloud computing platform (TCCP) for ensuring the confidentiality

and integrity of computations that are outsourced to IaaS services. The TCCP provides the abstraction of a closed box execution environment for a customer's VM, guaranteeing that no cloud provider's privileged administrator can inspect or tamper with its content.

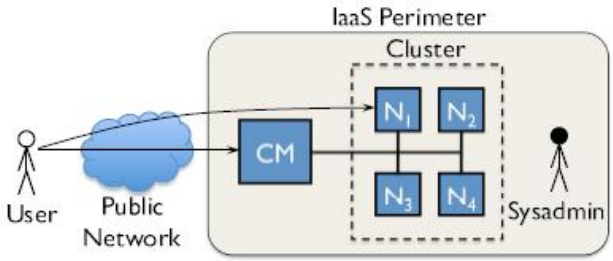

Figure 4: Simplified Architecture of Eucalyptus

Moreover, before requesting the service to launch a VM, (fig 2.2) above shows the TCCP allows a customer to reliably and remotely determine whether the service backend is running a trusted TCCP implementation. This capability extends the notion of attestation to the entire service, and thus allows a customer to verify if its computation will run securely.

In this research paper we show how to leverage the advances of trusted computing technologies to design the TCCP. Section 2 introduces these technologies and describes the architecture of an IaaS service. Section 3 presents our design of TCCP. Although we do not yet have a working prototype of TCCP, the design is sufficiently detailed that we are confident that a solution to the problem under discussion is possible. The Trusted Computing Group (TCG) [10] proposed a set of hardware and software technologies to enable the construction of trusted platforms. In particular, the TCG proposed a standard for the design of the 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 cryptographic functions that cannot be modified. The respective manufacturers sign the corresponding public key to guarantee the correctness of the chip and validity of the key.

Trusted platforms [1, 4, 5, and 9] leverage the features of TPM chips to enable remote attestation. This mechanism works as follows. At boot time, the host computes a measurement list ML consisting of a sequence of hashes of the software involved in the boot sequence, namely the BIOS, the boot loader, 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. The platform asks the local TPM to create a message containing both the ML and the, 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 nonce's match and the ML correspond to a configuration it deems trusted, a remote party can reliably identify the platform on a UN rusted host. A trusted platform like Terra [4] implements a thin VMM that enforces a closed box execution environment, meaning that a guest VM running on top cannot be inspected or modified by a user with full privileges over the host. The VMM guarantees its own integrity until the machine reboots. Thus, a remote party can attest to the platform running at the host to verify that a trusted VMM implementation is running, and thus make sure that her computation running in a guest VM is secure. Given that a traditional trusted platform can secure the computation on a single host, a natural approach to secure an IaaS service would be to deploy the platform at each node of the service's backend. However, this approach is insufficient: a sys admin can divert a customer's VM to a node not running the platform, either when the VM is launched (by manipulating the CM), or during the VM execution (using migration).

In this research paper, we argue that concerns about the confidentiality and integrity of their data and computation are a major deterrent for enterprises looking to embrace cloud computing. We present the design of a trusted cloud computing platform (TCCP) that enables IaaS services such as Amazon EC2 to provide a closed box execution environment. TCCP guarantees confidential execution of guest VMs, and allows users to attest to the IaaS provider and determine if the service is secure before they launch their VMs. We plan to implement a fully functional prototype based on our design and evaluate its performance in the near future.

Tamper Detection In Audit Logs

Audit logs are considered good practice for business systems, and are required by federal regulations for secure systems, drug approval data, medical information disclosure, financial records, and electronic voting. Given the central role of audit logs, it is critical that they are correct and inalterable. It is not sufficient to say, \our data is correct, because we store all interactions in a separate audit log." The integrity of the audit log itself

must also be guaranteed. This paper proposes mechanisms within a database management system (DBMS), based on cryptographically strong one-way hash functions, that prevent an intruder, including an auditor or an employee or even an unknown bug within the DBMS itself, from silently corrupting the audit log. We propose that the DBMS store additional information in the database to enable a separate audit log validate to examine the database along with this extra information and state conclusively whether the audit log has been compromised. We show with an implementation on a high-performance storage engine that the overhead for auditing is low and that the validator can efficiently and correctly determine.

Operating systems and databases. We address each in turn. Mercury raises the need to audit the audit log [14]. Peha [17] uses, as we do, one-way hash functions and a \trusted" notary to hash and store every transaction. Our approach tiers in that we make no assumptions about the DBMS, or even the hardware it executes on, remaining in the trusted computing base following an intrusion; Peha on the other hand advocates a "notary on a chip". Unlike Peha, we integrate hashing with stamping of tuples in the table, and we consider system issues such as the need to hash tuples and using partial result authentication codes to link transactions. Peha simply batches transactions together by hashing all the data in all the transactions, which will undoubtedly result in very poor performance, as we discussed in detail in Section 6.1. Peha goes into more detail on how customers, notarizes, validated, and auditors can use public key encryption to coordinate. Note that since we send the notarization service only hash values, no private data that is revealed to that external service. It may still be useful to encrypt the tuples that own from the database to the validate, if that process communicates with the DBMS over no secure channels.

As mentioned, Schneider and Kelsey address audit logs that are used for later forensic investigations into detected intrusions [10]. Their requirements tier considerably from ours. In particular, they render the log entries impossible for the attacker to read. They use a hash linking in a similar way to our algorithm. They do not consider efficiency issues, which are critical in our situation where an online transactional database is being logged. Markel proposed a digital signature system based on a secure conventional encryption function over a tree of document from utilized within a notarization service, but is not directly applicable to our problem of hashing the data of individual transactions. Divan et al. applied the Merle Tree authentication mechanism to both relational [6] and XML [5] data. Here the model is deferent queries over static data which has been previous digested are evaluated by an insecure server The query results are sent to clients, which can independently verify, using the digest, that the result contains all the requested records and no superiors records.

Existing Audit Log Techniques

The traditional way to protect logging data from tampering is to write it to an append-only device, such as a Write Once Read Multiple (WORM) optical drive or a continuous-feed printer. The security of such schemes assumes, however, that the computing site will not be compromised. If this is a possible attack scenario the logging data can be sent to a remote site over the network, so called remote logging. Log replication can be used to send the data to several hosts to require the attacker to physically compromise several sites. Schneider and Kelsey [11] describe a secure audit log system. The idea is roughly as follows.

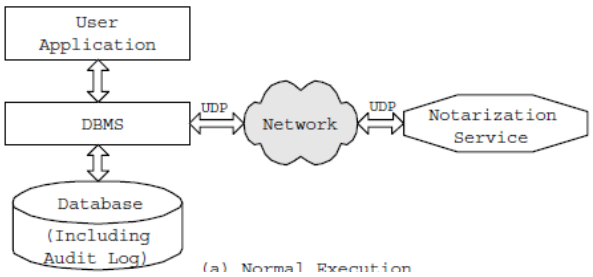

Figure 5: Normal Execution

An untrusted machine U (on which the log is kept) initially shares a secret authentication key A0 with a trusted machine T . To add the j:th log entry Dj , U computes K = hash(Aj) (an encryption key), C = EK(Dj) (the encrypted log entry), Yj = hash(Yj□1;C) (the j:th entry in a chain of hashes, where Y□1 = 0), and Zj =MACAj (Yj) (a keyed hash (Message Authentication Code) of Yj). Then the j:th entry hC; Yj ;Zj i is written to the log, a new authentication key Aj+1 = hash(Aj) is constructed, and Aj is destroyed. An attacker who compromises U at time t can delete (but not read nor modify) any of the first log entries, since he will only have access to At+1 but not to any of the previous A0 : : :At.

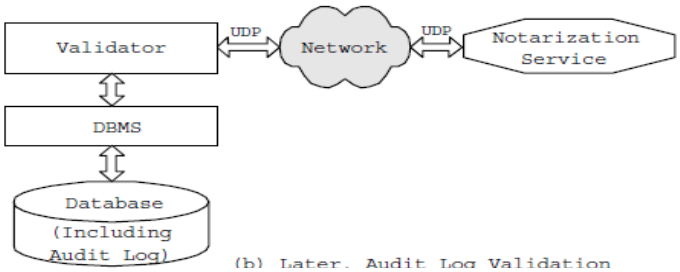

Figure 6: Later, Audit Log Validation

In the figure shown above, while there is no way to prevent the attacker from deleting some or all of the log entries (or appending his own entries), any such attempted tampering will be detected by T on its next interaction with U. Furthermore, since each log entry is encrypted with a key derived from A0 (which is only stored permanently on T) the attacker cannot read past log entries to and out if his attack was noticed or not. As applications require access to the database (and often read access to the audit log), these existing techniques are not applicable to tamper detection of transactional database audit logs.

APEL: An Implementation Of Grid Accounting Using R-GMA

APEL (Accounting Processor for Event Logs) parses batch, system and gatekeeper logs generated by a site and builds accounting records, which provide a summary of the resources consumed based on attributes such as CPU time, Wall Clock Time, Memory and grid user DN. [12] The accounting data is published into the R-GMA information and monitoring system, and archived for processing by a graphical front-end utilized by the accounting web tool. Apel is a log processing application which is use to interpret gatekeeper and batch system logs to produce accounting records. It currently supports PBS and LSF batch systems but can easily be extended to support other variants. This is possible because of a newly developed plug-in architecture which separates the core functionality from the actual log parsing. Apel provides a plug-in which parses PBS and LSF batch systems logs.

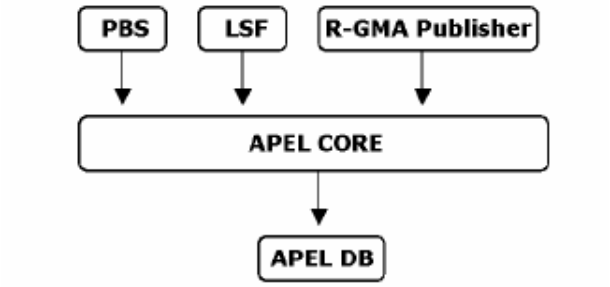

Figure 7: APEL

A plug-in exists to publish accounting records into RGMA. Each plug-in connects to the underlying DB via the Apel core. A complete accounting record is composed of (among others) the grid user, the job id of the submitted job and the resources used when executing the job. This information is typically dispersed between several different log file types such as those produced by the gate keeper or batch system. For resource usage, a query is issued to the site's GIIS to lookup the CPU performance for the computing nodes where the job was executed. Apel attempts to collect all this piecemeal information together and manages it within a database. A further process carried out by Apel then attempts to join the data together to produce a list of final accounting records with all necessary details filled-in. Apel is then used to publish the generated accounting records into R-GMA whereby they are collated at the GOC using an R-GMA secondary producer, as shown in the above figure. Apel provides support for republishing the complete local copy of accounting records to RGMA (in cases when the GOC was offline). It also provides a mechanism for reliable delivery using a basic integrity check to compute the number of records that were last published compared with the actual count stored on the GOC.

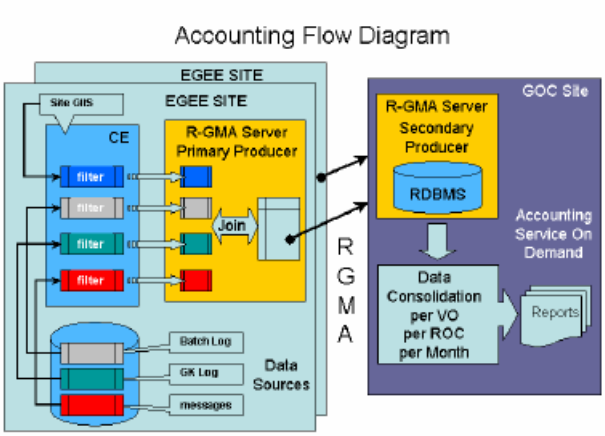

Figure 8: Accounting flow diagram providing a global overview of the data collection process (APEL), and the web reporting service.

Each accounting record is unique and there is only one record per grid job. The records may be consolidated in different ways to provide high-level views of accounting data, such as the total CPU time consumed for each LHC VO at the tier-1 computing centers. [13]

Condor-G

In recent years, there has been a dramatic increase in the amount of available computing and storage resources, yet few have been able to exploit these resources in an aggregated form. We present the Condor-G system, which leverages software from Globes and Condor to allow users to harness multi-domain resources as if they all belong to one personal domain. We describe the structure of Condor-G and how it handles job management, resource selection, and security and fault tolerance. The Condor-G system leverages the significant advances that have been achieved in recent years in two distinct areas: security, resource discovery, and resources access in multi-domain environments, as supported within the Globus Toolkit [14], and management of computation and harnessing of resources within a single administrative domain, specifically within the Condor system [15,16]. In brief, we combine the inter-domain resource management protocols of the Globus Toolkit and the intra-domain resource management methods of Condor to allow the user to harness multi-domain resources as if they all belong to one personal domain. The user defines the tasks to be executed; Condor-G handles all aspects of discovering and acquiring appropriate resources, regardless of their location; initiating, monitoring, and managing execution on those resources; detecting and responding to failure; and Notifying the user of termination. The result is a powerful tool for managing a variety of parallel computations in Grid environments.

Condor-G's utility has been demonstrated via record setting computations. For example, in one recent computation a Condor-G agent managed a mix of desktop workstations, commodity clusters, and supercomputer processors at ten sites to solve a previously open problem in numerical optimization. In this computation, over 95,000 CPU hours were delivered over a period of less than seven days, with an average of 653 processors being active at any one time. In another case, resources at three sites were used to simulate and reconstruct 50,000 high-energy physics events, consuming 1200 CPU hours in less than a day and a half. In the rest of this article, we describe the specific problem we seek to solve with Condor-G, the Condor-G architecture, and the results obtained to date.

GRASP: A Grid Resource Allocation System Based On OGSA

In this research paper, we describe GRASP, a grid resource allocation system based on OGSA. In order to submit job to the grid resources in more efficient and convenient manner, we support some features for user-friendly resource allocation such as resource brokering, scheduling, monitoring, and so forth. GRASP supports any scientific applications with the high performance computing features such as MPI and applications with high throughput computing features such as parameter studies.

Secure Virtual Machine Execution Under An Untrusted Management OS

Virtualization is a rapidly evolving technology that can be used to provide a range of benefits to computing systems, including improved resource utilization, software portability, and reliability. For security-critical applications, it is highly desirable to have a small trusted computing base (TCB), since it minimizes the surface of attacks that could jeopardize the security of the entire system. In traditional virtualization architectures, the TCB for an application includes not only the hardware and the virtual machine monitor

(VMM), but also the whole management operating system (OS) that contains the device drivers and virtual machine (VM) management functionality. For many applications, it is not acceptable to trust this management OS, due to its large code base and abundance of vulnerabilities. [17]

In this research paper, we address the problem of providing a secure execution environment on a virtualized computing platform under the assumption of an untrusted management OS. We propose a secure virtualization architecture that provides a secure run-time environment, network interface, and secondary storage for a guest VM. The proposed architecture significantly reduces the TCB of security-critical guest VMs, leading to improved security in an untrusted management environment. We have implemented a prototype of the proposed approach using the Xen virtualization system, and demonstrated how it can be used to facilitate secure remote computing services. We evaluate the performance penalties incurred by the proposed architecture, and demonstrate that the penalties are minimal.

VTPM: Virtualizing The Trusted Platform Module

We present the design and implementation of a system that enables trusted computing for an unlimited number of virtual machines on a single hardware platform. To this end, we virtualized the Trusted Platform Module (TPM). As a result, the TPM's secure storage and cryptographic functions are available to operating systems and applications running in virtual machines. Our new facility supports higher-level services for establishing trust in virtualized environments, for example remote attestation of software integrity. We implemented the full TPM specification in software and added functions to create and destroy virtual TPM instances. We integrated our software TPM into a hypervisor environment to make TPM functions available to virtual machines. Our virtual TPM supports suspend and resume operations, as well as migration of a virtual TPM instance with its respective virtual machine across platforms. We present four designs for certificate chains to link the virtual TPM to hardware TPM, with security vs. efficiency trade-offs based on threat models. Finally, we demonstrate a working system by layering an existing integrity measurement application on top of our virtual TPM facility.

TISA: Toward Trustworthy Services In A Service-Oriented Architecture

Verifying whether a service implementation is conforming to its service-level agreements is important to inspire confidence in services in a service-oriented architecture (SoA). Functional agreements can be checked by observing the published interface of the service, but other agreements that are more non-functional in nature, are often verified by deploying a monitor that observes the execution of the service implementation. A problem is that such a monitor must execute in an untrusted environment. Thus, integrity of the results reported by such a monitor crucially depends on its integrity. [18]

We contribute an extension of the traditional SOA, based on hardware-based root of trust that allows clients, brokers and providers to negotiate and validate the integrity of requirements monitor executing in an untrusted environment. We make two basic claims: first, that it is feasible to realize our approach using existing hardware and software solutions, and second, that integrity verification can be done at a relatively small overhead. To evaluate feasibility, we have realized our approach using current software and hardware solutions. To measure overhead, we have conducted a case study using a collection of Web service implementations available with Apache Axis implementation.

Inspector Gadget

Inspector Gadget focuses on a specific, but important, class of applications: distributed data processing. In that context, IG aims to enable a wide variety of behaviors (most of the ones in Table 1) with simple coding, and to avoid intrusive modifications to the underlying dataflow system and the data it manages. We believe those goals largely set it apart from other work. That said, there are several prior projects that do overlap in some ways with ours. Many of them focus on a narrower set of behaviors (e.g. just forward tracing, latency profiling and overhead profiling) and embed in the underlying systems at a much lower level, thereby potentially achieving better performance (for those behaviors) at the expense of more intrusiveness. One class of mechanisms for achieving a few (4/14) of the behaviors is tainting with tracing [19, 20]. These approaches annotate data with special markers that enable it to be tracked as it moves through complex system(s). Perhaps the most relevant of these is X-trace [21], which can track data in and between nodes of distributed systems, including dataflow systems such as Hadoop. Aguilera et al. [22] focuses on network traces, and seeks to identify causal relationships and measure latency for chains of RPC calls. It makes a simplifying assumption that latency is mostly due to the network. Inspector Gadget is also interested in latency and causality, but because our dataflow programs can be computation and I/O intensive, we cannot rely on such an assumption. Instead, IG benefits from a different kind of simplifying assumption: the set of possible dataflow operators and control flow situations is small and known a priori.

SYSTEM ANALYSIS

Existing System

The billing system with limited security concerns and the micropayment-based billing system require a relatively low level of computational complexity. The average billing latency for billing system with limited security is 4.06ms, for micro payment-based billing system is 4.07ms. Nevertheless, these systems are inadequate in terms of transaction integrity, non-repudiation and trusted SLA monitoring.

In spite of the consensus that PKI-based billing system offers a high level of security through two security functions excluding trustworthy SLA monitoring, the security comes at the price of extremely complex PKI operations with the average billing latency of 82.51ms. The micropayment-based schemes such as Pay Word, MiniPay, e-coupons, and Net Pay. Consequently, when a PKI-based billing system is used in cloud computing environment the high computational complexity causes high deployment cost and high operational overload because the PKI operations must be performed by the user and the CSP.

Proposed System: In this research paper, we propose a secure and no obstructive billing system called THEMIS as a remedy for these limitations. The system uses a novel concept of a cloud notary authority for the supervision of billing. The cloud notary authority generates mutually verifiable binding information that can be used to resolve future disputes between a user and a cloud service provider in a computationally efficient way. This project will produce the secure billing through monitoring the service level agreement (SLA) by using the SMon module. CNA can get a service logs from SMon and stored it in a local repository for further reference. Even administrator of a cloud system cannot modify or falsify the data.

Adavantages In Proposed System

- Billing transactions are non obstructive.
- Minimal Computation Cost.
- Trusted Service level agreement (SLA) monitoring by SMon.
- Minimum transaction latency.

Problem Definition

For the billing transaction existing system used public key infrastructure (PKI)-based digital signature into each billing transaction to prevent corruption. Several studies have addressed this issue by deploying a PKI-based digital signature mechanism in an underlying security layer; however, they were handicapped by computational overhead due to the extreme complexity of the PKI operations. Consequently, when a PKI-based billing system is used in a cloud computing environment, the high computational complexity causes high deployment costs and a high operational overhead because the PKI operations must be performed by the user and the CSP.

Modules

- User Interface Design
- Cloud Service Provider
- User
- Cloud Notary Authority (CNA)
- Monitor
- Action against SLA violation

Module Description

User Interface Design: User Interface Design have a purpose that a user to move from login page to user page of the website. In this we want to enter our user name and password provided by Service provider as in the below figure shown.

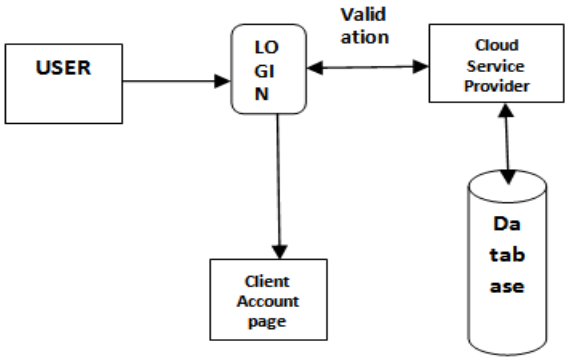

Figure 9: User Interface Design

If we enter the valid password and user name then only the user can move login page to user window while entering user name and password it will check username and password is match or not. If we enter any wrong username or wrong password it generates some error message. So we are preventing from unauthorized user entering into the service provider website. It will provide a good security for our project. So Service provider contain user name and password server also check the authentication of the user. It will improve the security and preventing from unauthorized user enters into the website. In our project we are using java swings for creating design. Here we are validating the users who are going to access the Service providers.

Cloud Service Provider

Service provider has a job of providing a service like software to the cloud users. In our proposed method, CSP doesn't provide billing transaction to the user. It is due to the reason if billing transaction performed in the CSP then complexity in security to be provided for billing transaction increases the overhead. If the user logged in for service, CSP validate the user whether he\she is an authenticated user or not as shown in the figure below.

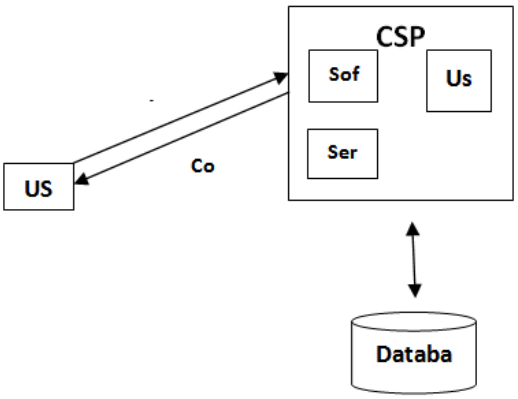

Figure 10: Cloud Service Provider

Once if user is found authenticated user then it waits for service check in message else it found any unauthenticated user it will send the error message. If it received the service check in message then it responds the user by transmitting the agreement and hash chain (one time key). After getting the service request from the user, CSP provide the requested service to the user. It is also have a contact with the Cloud notary authority. It will provide the service until it receive the service checkout message. The CSP enables users to scale their capacity upwards or downwards regarding their computing requirements and to pay only for the capacity that they actually use.[23]

USER: User can access a service from the Cloud Service Provider by authenticated login process. We assume that users are thin clients who use services in the cloud computing environment. To start a service session in such an environment, each user makes a service check-in request to the CSP with a billing transaction. To end the service session, the user can make a service check-out request to the CSP with a billing transaction. Once if the users send the service check-in message it can get the contract from the CSP.

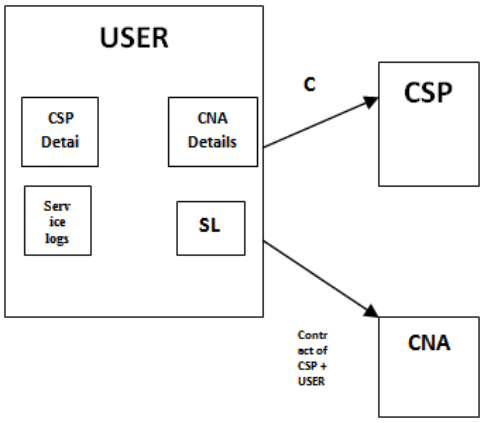

Figure 11: USER

In the above figure after receiving the one time keywords in the contract it can be able to access the service from the CSP. Now user log details are stored in Monitor for future disputes. After accessing the service, user want billing transaction. If he\she wants the bill means it should send the contract of the CSP with contract of the user to the CNA. The details checked by the CNA are identical then user can receive the bill binding information along with confirmation message. If any error occurred or forgery activity found from the user side then the user will receive the penalty for that.

Cloud Notary Authority (CNA)

Cloud Notary Authority acts as a THEMIS in our cloud billing transaction. He is an authority to generate the billing transaction for the cloud service. [24] The CNA provides a mutually verifiable integrity mechanism that combats the malicious behavior of users or the CSP. The process, which involves a generation of mutually verifiable binding information among all the involved entities on the basis of a one-way hash chain (One time key), is computationally efficient for a user and the CSP.

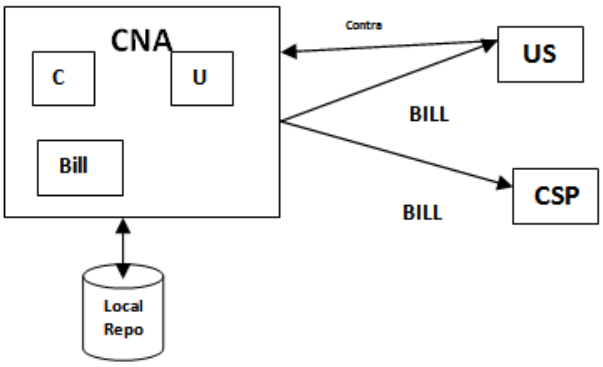

Figure 12: Cloud Notary Authority

If user wants billing for the service then it sends the contract of the user and contract of CSP to the CNA. In CNA it checks both the contract; if it is found as identical then it generates the bill as binding information and sends the confirmation message to the user and the CSP. If it is not identical then it receives the log details from the monitor. If forgery found at user side it sends the penalty to the user. If it found at CSP side it cancels the payment to the CSP. CNA provide the billing transaction which can be verifiable and also forgery resistive in cloud environment.

Monitor

Monitor is a module which continuously monitors all the log activities of the CSP and the user. For monitoring it uses a technique called S-Mon. The S-Mon has a forgery-resistive SLA measuring and logging mechanism, which enables it to monitor SLA violations and take corrective actions in a trusted manner. After the service session is finished, the data logged by S-Mon are delivered to the CNA. In figure 3.5.5 shows

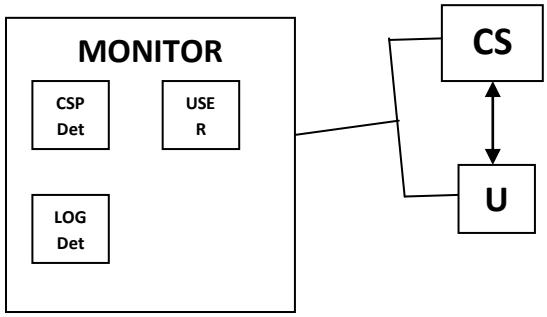

Figure `13: Monitor

We devised S-Mon in such a way that it can be deployed as an SLA monitoring module in the computing resources of the user. Once SLA has been violated S-Mon sends all the log details to the CNA. After verifying the log details CNA perform further action. Monitor has a local repository for storing all the log details of the user to monitor the SLA for the future disputes. So it can be verifiable in future too. Here monitor plays important role against billing transaction forgery which leads to forgery resistive billing transaction.[25]

Action Against SLA Violation

Once the CNA found forgery from cloud services it can't directly take any action against them without knowing the reason. At that time it sends the message to Monitor to send the all log details about the transaction. Once it receives the log message from the monitor it compares the contract and the log details. In the fig 3.5.6 once the forgery found from CSP side it cancels the payment to the CSP and sends the message to the CSP. If it found from the user side it assign penalty to the user according to the severity of the forgery from the user side and sends the message to the user. CNA also maintains the local repository after the action taken against the SLA violation.

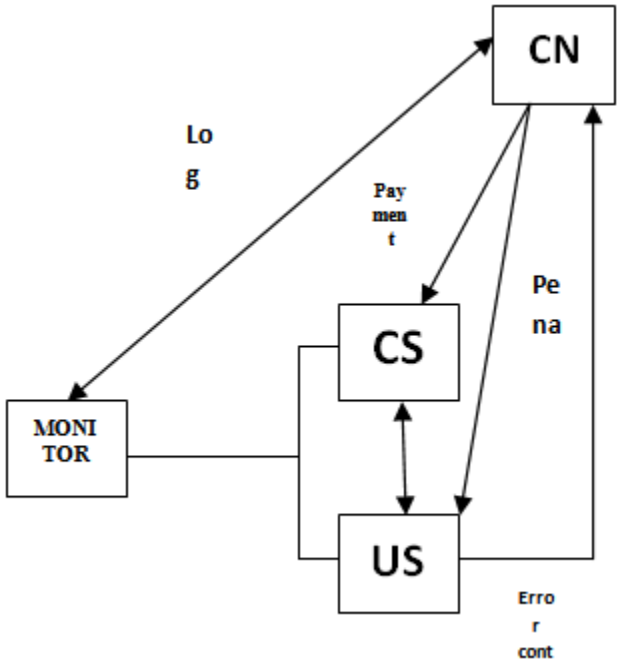

Figure 14: Action against SLA violation

Given Input Expected Output

User Interface Design

Input:

- Registration
- Username
- Password

Output:

- Registered in Database
- Login successfully
- Open client home page

Cloud Service Provider

Input:

- Check-in message
- User response with one time key

Output:

- Contract including SLA and Hash chain
- Software Service

USER

Input:

- Contract and Hash chain from CSP
- Access software from CSP

Output:

- Response to csp with Hash chain
- Send contract of user and CSP to CAN

Cloud Notary Authority (Cna)

Input:

- contract of user and CSP from user

Output:

- Generation of billing transaction
- Send confirmation message to CSP and user

MONITOR

Input:

- Logging details of CSP
- Logging details of user

Output:

- Stored in repository
- Send logging details to the CNA when error occurred in contract

Action Against Sla Violation

Input:

- Logging details of user and CSP from monitor

Output:

- Cancel payment for CSP
- Provide penalty for user
- Maintain local repository for future dispute

SYSTEM DESIGN & IMPLEMENTATION

Usecase Diagram For Modules

A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.

Usecase

A defined purposeful, interaction between a system & a human or non human actor that is playing a specify role outside the system. In the below fig 4.2 shows the detail description.

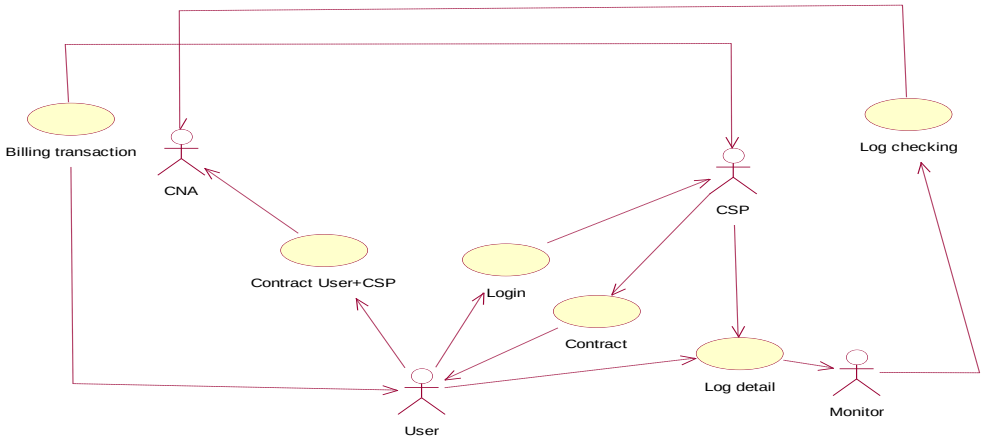

Figure 15: Use case for User and CAN

From the above use case diagram we can know that every actor have some action with other actor in the network. User have log in relation with CSP. For that CSP response with contract contains one time key. For billing transaction USER send the contract of both user and CSP to the CNA. Then CNA produce Billing transaction and also send the confirmation message to both user and CSP. All the logging details are stored by monitor which will be send to the CNA for disputes.

Activity Diagram

Activity diagram are a loosely defined diagram to show workflows of stepwise activities and actions, with support for choice, iteration and concurrency.

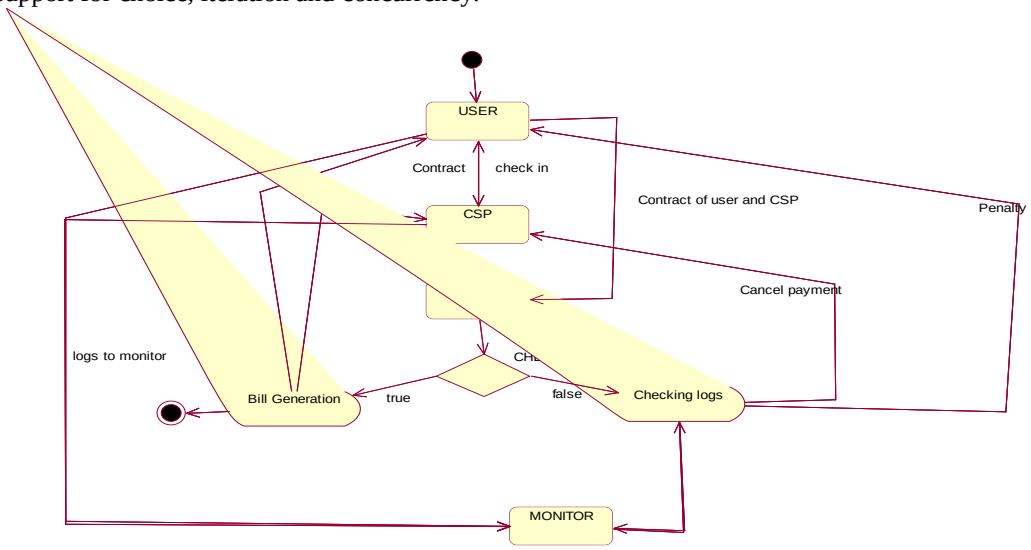

Figure 16: Active Diagrams for User and CNA

In the above figure, it shows a description UML, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. UML activity diagrams could potentially model the internal logic of a complex operation. In many ways UML activity diagrams are the object-oriented equivalent of flow charts and data flow diagrams (DFDs) from structural development.

In this activity diagram user can access service from the CSP by sending the check in message. After accessing the service if user wants billing transaction it sends the contract of the user and CSP to authority. Authority checks both contract if it is identical then it generates the bill and send to both the user and csp. If it found any mismatch then it checks the log details from the monitor and sends the action against the CSP/user.

Sequence Diagram

A sequence diagram in UML is a kind of interaction diagram that shows how processes operate with one another and in what order. A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams typically are associated with use case realizations in the Logical View of the system under development. Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagrams.

From this sequence diagram it is shown that the sequence from user to can for bill generation. First user get a contract by check in message and access the service using that contract. After accessing the service it request the CNA for bill generation. CNA checks the both contract if it is identical then it generates the bill and send the information to the user and csp. If it is not identical then it checks the log details from monitor and take action against the SLA violator.In the figure below it is shown in detail about sequence diagram.

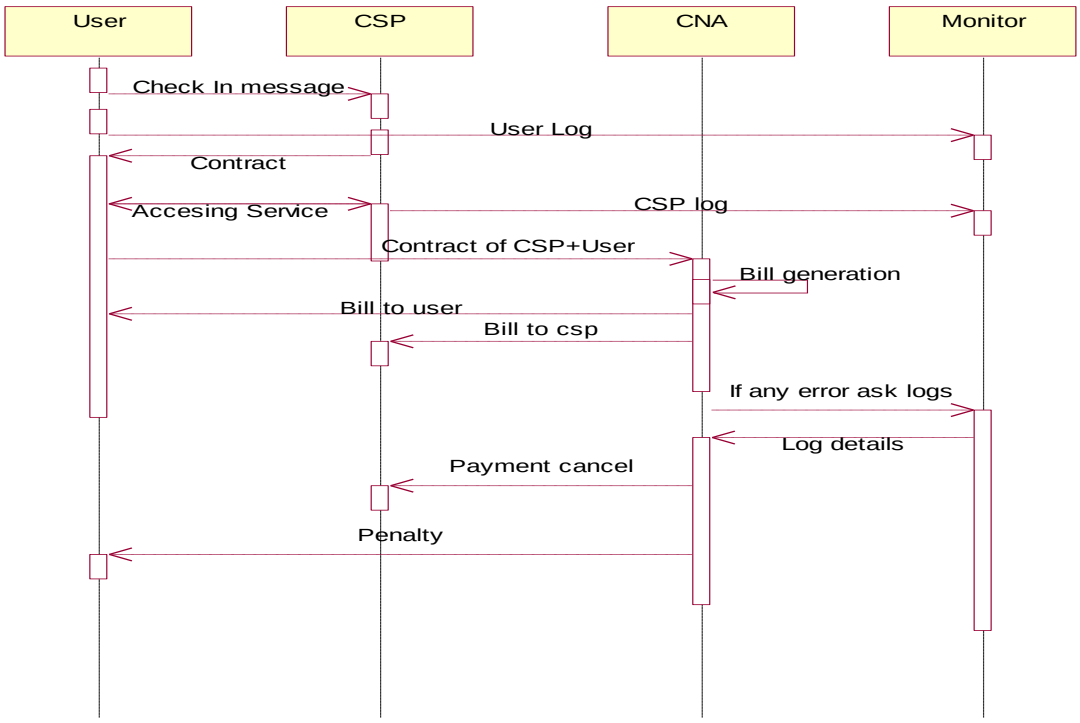

Figure 17: Sequence diagram for User and CSP

Collaboration Diagram

A collaboration diagram show the objects and relationships involved in an interaction, and the sequence of messages exchanged among the objects during the interaction. The collaboration diagram can be a decomposition of a class, class diagram, or part of a class diagram. it can be the decomposition of a use case, use case diagram, or part of a use case diagram. The collaboration diagram shows messages being sent between classes and object (instances). A diagram is created for each system operation that relates to the current development cycle (iteration).

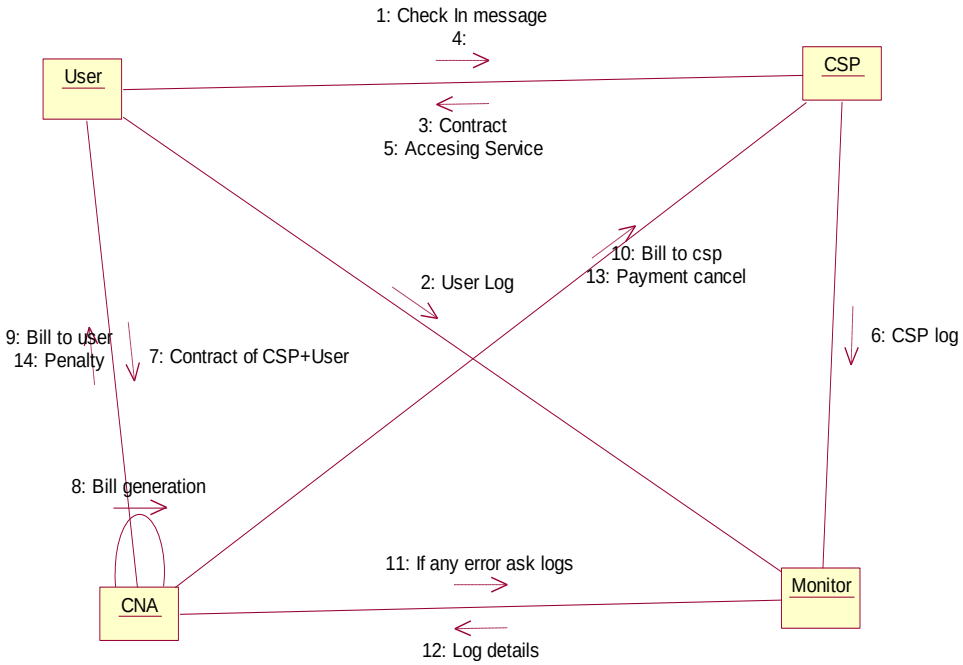

Figure 18: Collaboration diagram for user

In the above Collaboration diagram, it shows how the Can takes the dyanamic action like if it founds the contract are identical then it generates the bill or else it checks the log details from the moniotor and take the action against the violator. So this type of dynamic flow and astatic flow like user validation and response activity are represented by using this diagram.

Class Diagram

A class diagram in the UML is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, and the relationships between the classes. Private visibility hides information from anything outside the class partition. Public visibility allows all other classes to view the marked information. Protected visibility allows child classes to access information they inherited from a parent class. In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among the classes. A class diagram is an illustration of the relationships and source code dependencies among classes in the Unified Modeling Language (UML). In a class diagram, the classes are arranged in groups that share common characteristics. A class diagram resembles a flowchart in which classes are portrayed as boxes, each box having three rectangles inside. The top rectangle contains the name of the class; the middle rectangle contains the attributes of the class; the lower rectangle contains the methods, also called operations, of the class. Lines, which may have arrows at one or both ends, connect the boxes. In the below figure, it shows detail description these lines define the relationships, also called associations, between the classes.

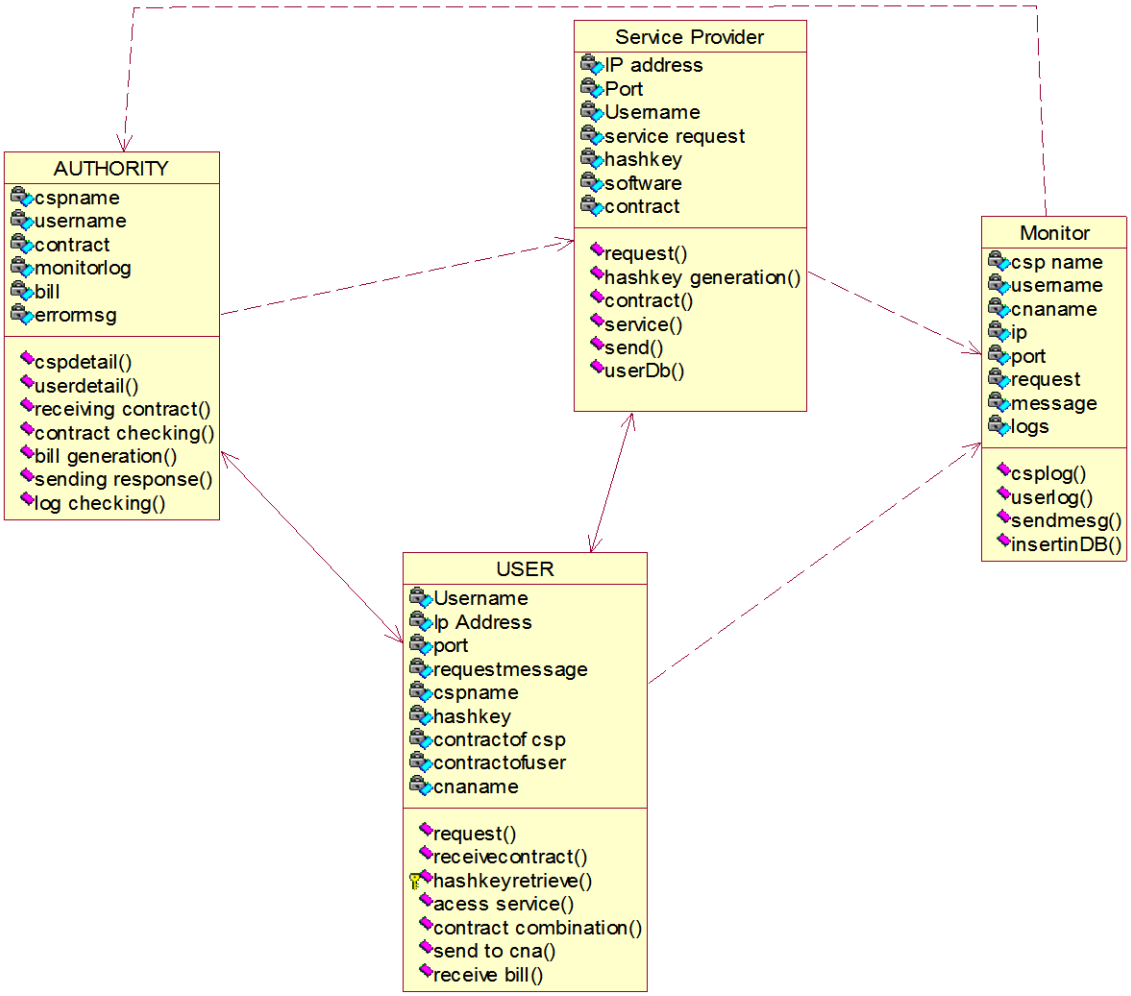

Figure 19: Class Diagram

From this class diagram how the classes are interconnected to perform the action is explained. Service provider class is contact with the user class for validating the user and sends the contract. After that user class have send the request to the CNA class for bill generation and it calls the both service provider and user class for confirmation. After finishing it receives message from monitor for checking log details.

Data Folw Diagram

A data flow diagram (DFD) is a graphical representation of the "flow" of data through an information system. It differs from the flowchart as it shows the data flow instead of the control flow of the program. A data flow diagram can also be used for the visualization of data processing. The DFD is designed to show how a system is divided into smaller portions and to highlight the flow of data between those parts.

Data Flow Diagram (DFD) is an important technique for modeling a system's high-level detail by showing how input data is transformed to output results through a sequence of functional transformations. DFDs reveal relationships among and between the various components in a program or system. In the below figure, DFD diagram consists of four major components: entities, processes, data stores and data flow.

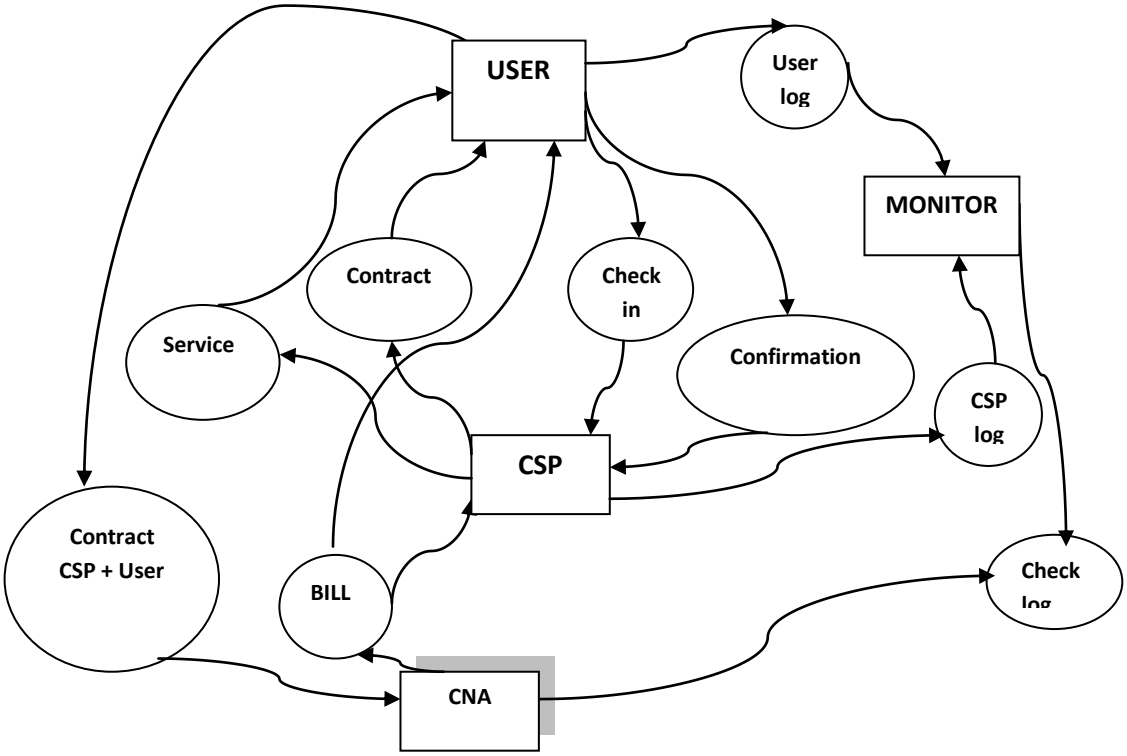

Figure 20: Data Flow Diagram

From DFD 0 the flow from user to csp is shown. User requests the CSP and gets the service from the CSP after giving confirmation with hash chain. Monitor has the log details of both monitor and the CSP from DFD 1 we can know that after accessing the service from CSP, user sends the contract of user and CSP to the CNA. CNA Verify both the contracts if both are identical then it generates the bill and sends to both user and CSP. If not it checks the log details at the monitor and take further action against the violator.

SYSTEM ARCHITECTURE

Architecture diagram shows the relationship between different components of system. This diagram is very important to understand the overall concept of system. Architecture diagram is a diagram of a system, in which the principal parts or functions are represented by blocks connected by lines that show the relationships of the blocks. They are heavily used in the engineering world in hardware design, electronic design, software design, and process flow diagrams.

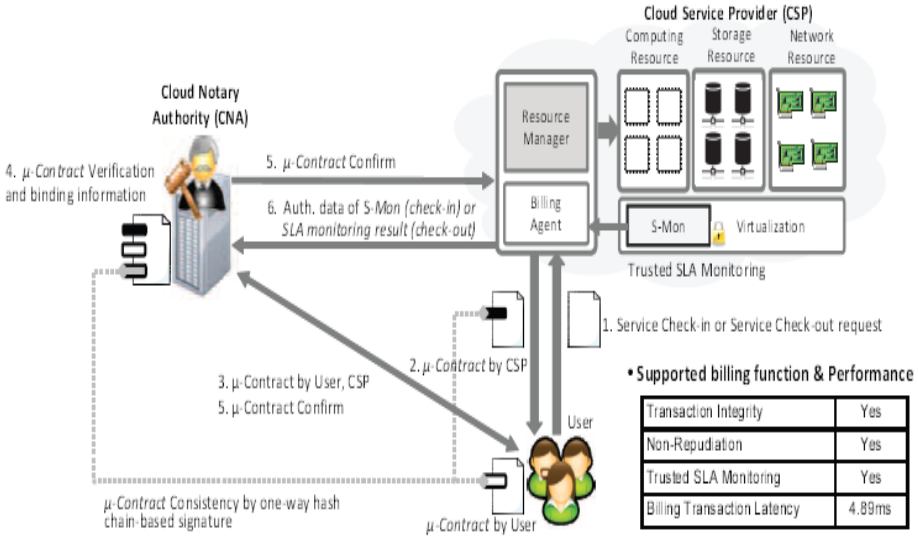

Figure 21: System Architecture Diagram

System Architecture Diagram shows that the entire flow of the project. When the users are validated by the CSP, it will send the contract with hash chain to the user. After that the user request for the service with that hash key. Once user finished accessing service from the CSP it sends the contract of the user and CSP to the authority. Authority checks the contract; if it is identical then it generates the bill and sends the confirmation message to the CSP and the user. If it found error it checks the log detail from the monitor and takes the action against the person who violates the service level agreement.

Object Diagram

An object diagram in the Unified Modeling Language (UML), is a diagram that shows a complete or partial view of the structure of a modeled system at a specific diagram focuses on some articular set of object instances and attributes, and the links between the instances. A correlated set of object diagrams provides insight into how an arbitrary view of a system is expected to evolve over time. Object diagrams are more concrete than class diagrams, and are often used to provide examples, or act as test cases for the class diagrams. Only those aspects of a model that are of current interest need be shown on an object diagram. An object diagram may be considered a special case of a class diagram.

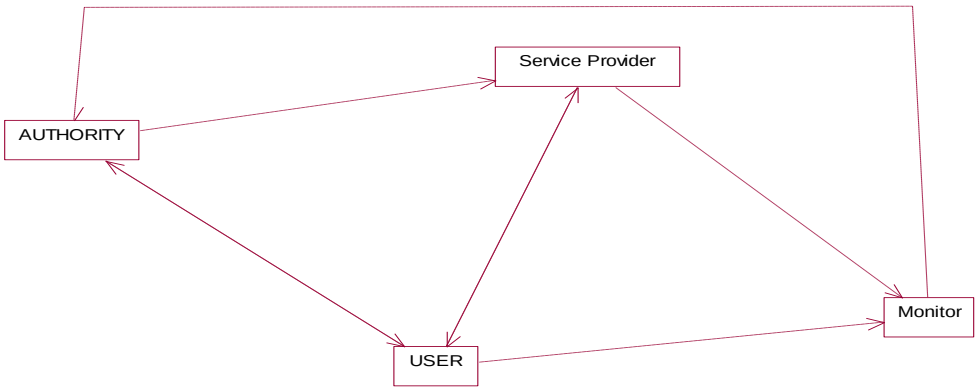

Figure 22: Object Diagram

From the figure above which is an object diagram, we can know that how we create a object for a class and how they perform the action. Simply it shows that flow of object. We create object for a class like service provider, authority, user and monitor to perform the action of bill generation in efficient way. It is a diagram that shows a complete or partial view of the structure of a modeled system.

State Diagram

A state diagram is a type of diagram used in computer science and related fields to describe the behavior of systems. State diagrams require that the system described is composed of a finite number of states; sometimes, this is indeed the case, while at other times this is a reasonable abstraction. There are many forms of state diagrams, which differ slightly and have different semantics. A state diagram, also called a state machine diagram or state chart diagram, is an illustration of the states an object can attain as well as the transitions between those states in the Unified Modeling Language (UML). A state diagram resembles a flowchart in which the initial state is represented by a large black dot and subsequent states are portrayed as boxes with rounded corners. There may be one or two horizontal lines through a box, dividing it into stacked sections.

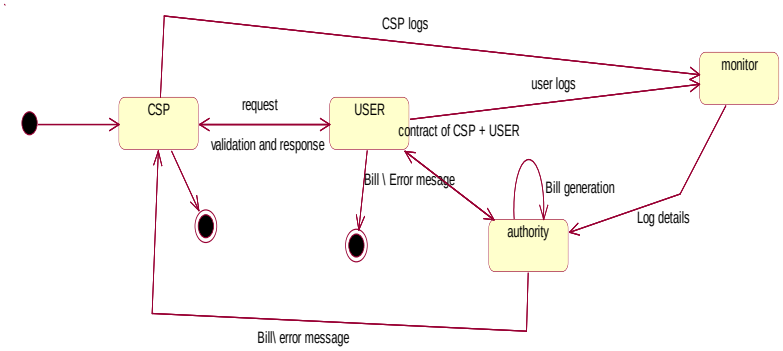

Figure 23: State Diagram

In that case, the above fig 4.10 the upper section contains the name of the state, the middle section (if any) contains the state variables and the lower section contains the actions performed in that state. If there are no horizontal lines through a box, only the name of the state is written inside it. External straight lines, each with an arrow at one end, connect various pairs of boxes. These lines define the transitions between states. The final state is portrayed as a large black dot with a circle around it. In this State diagram the process starts from validation of the user and then proceed with sending the contract. After accessing the service user send the contract to the authority. By checking the both contract authority will generate the bill. If it is error it checks the log details from the monitor and produce penalty or payment cancellation. After that it can send either bill or error message to botrh user and csp at that point process come to end.

Component Diagram

In the Unified Modeling Language, a component diagram depicts how components are wired together to form larger components and or software systems. They are used to illustrate the structure of arbitrarily complex systems. The Component Diagram helps to model the physical aspect of an Object-Oriented software system.

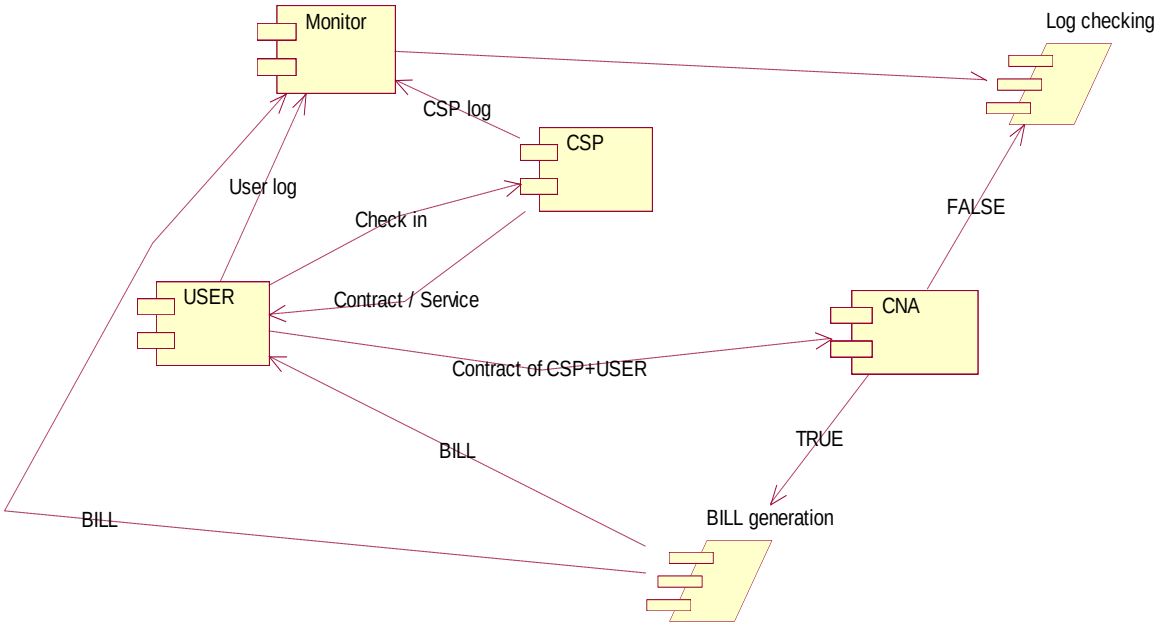

Figure 24: Component Diagram

It illustrates the architectures of the software components and the dependencies between them. Those software components including run-time components, executable components also the source code components. From this diagram components like cna, monitor, user, csp are connected to form the larger components. So can have further decision making dynamically.

E-R DIAGRAM

In software engineering, an entity-relationship model (ERM) is an abstract and conceptual representation of data. An entity-relationship (ER) diagram is a specialized graphic that illustrates the relationships between entities in a database. ER diagrams often use symbols to represent three different types of information. Boxes are commonly used to represent entities.

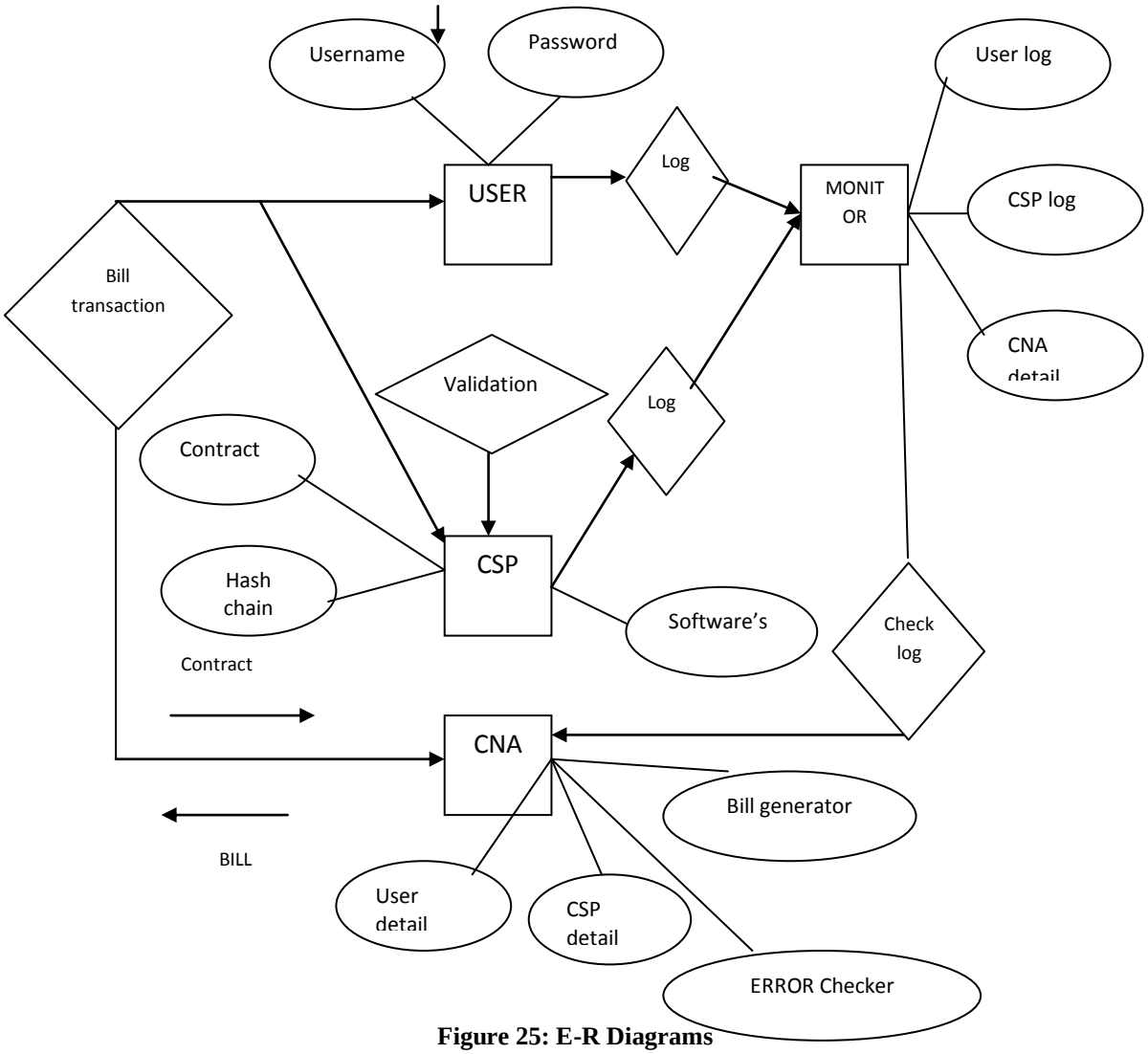

Figure 25: E-R Diagrams

Diamonds are normally used to represent relationships and ovals are used to represent attributes. E-R diagram shows in fig 4.12 that shows relationship between the various entities with attributes. In this diagram user have attributes like username and password having validation and service relationship with the entity CSP with the attributes contract.

II. Results & Conclusion

Our aim in this study was to provide a full-fledged trusted, non obstructive billing system tailored for a cloud computing environment. To accomplish this task, we thoroughly reviewed the ways in which existing billing systems are used in the environment. We consequently derived blueprints for THEMIS, our mutually verifiable, computationally efficient billing system. In addition to utilizing existing billing systems, we conceived and implemented the concepts of a CNA and S-Mon, which supervise billing transactions to make them more objective and acceptable to users and CSPs alike.

Our billing system features three remarkable achievements: First, we introduce a new concept of a CNA to ensure undeniable verification of any transaction between a cloud service user and a CSP. Second, our mutually verifiable billing protocol replaces prohibitively expensive PKI operations without compromising the security level of the PKI; as a result, it significantly reduces the billing transaction overhead. Last but not least, we devised a forgery-resistive SLA measuring and logging mechanism. By integrating the module into each cloud resource, we made the billing transactions more objective and acceptable to users and CSPs.

Applications And Future Enhancement

EPN MOBILE PHONE

This mobile phone have a application of transaction processing available at swiped rates through common smart phones, cell phones and PDA's. The ePN Mobile Credit Card, Check and Gift/Loyalty Application can prompt for invoice number, gratuity, other charges process the transaction real-time and show the transaction authorization number right on the phone display.

VOSS FULFILMENT SOLUTION

Specialty OSS vendors (Operational Support Systems) have developed end-to-end service orchestration solutions for service providers in the cloud communications space. VOSS Solutions is the leading OSS vendor in this public, cloud communications OSS space, with more tier-1 service provider customers than any other player.

Absolute Performance Sla Monitoring

Organizations have an increasing demand for business visibility. As a business executive, it is vital to know the state of your business-critical and revenue critical applications at all times. Knowing that your application is being managed to meet your business requirements is necessary to ensure 24x7 availability, transaction volume and performance of the application from the end-user perspective. Absolute Performance provides the visibility through custom SLA monitoring, enabling executives to view real-time SLA compliance and reporting, consolidated into a cohesive, easy to use portal view.

Future Enhancement

In future, the deployment of THEMIS in the context of existing cloud computing services requires minimal modification to the CSPs, CNA and users if seeking to provide mutually verifiable billing transactions. Our next step is to consider the scalability and fault tolerance of THEMIS. This fault tolerance can be implemented by Banking service.

References

[1]. Ki-Woong Park, Member, IEEE, Jaesun Han, Member, IEEE, JaeWoong Chung, Member, IEEE, and Kyu Ho Park, Member, IEEE Transactions on information forensics and security, Vol. 6, No. 2, JUNE 2012

[2]. Microsoft, "Microsoft, windows azure platform," 2010. [Online].Available: http://www.microsoft.com/windowsazure

[3]. M. Armbrust and A. E. Fox, "Above the clouds: A Berkeley view of cloud computing," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28, Feb 2009.

[4]. N. Santos, K. P. Gummadi, and R. Rodrigues, "Towards trusted cloud computing," in Proc. of USENIX HotCloud 2009.

[5]. R. T. Snodgrass, S. S. Yao, and C. Collberg, "Tamper detection in audit logs," in Proc. of 30th intl. conf. on Very large data bases, ser. VLDB '04. VLDB Endowment, 2004, pp. 504–515.

[6]. L. C. M. C. Rob Byrom, Roney Cordenonsib, "Apel: An implementation of grid accounting using r-gma," UK e-Science All Hands Conference, Nottingham, September 2005.

[7]. Frey, Tannenbaum, Livny, Foster, and Tuecke, "Condor-g: A computation management agent for multi-institutional grids," Cluster Computing, vol. 5, pp. 237–246, 2002.

[8]. O.-K. Kwon, J. Hahm, S. Kim, and J. Lee, "Grasp: A grid resource allocation system based on ogsa," in Proc. of the 13th IEEE Intl. Sympo. on High Performance Distributed Computing.IEEE Computer Society, 2004, pp. 278–279.

[9]. I. P. Release, "Tivoli: Usage and accounting manager," IBM press 2009.

[10]. P. working group, "http://www.ietf.org/html.charters/pkixcharter.html," 2008.

[11]. A. Guarise, R. Piro, and Werbrouck, "A. datagrid accounting system - architecture - v1.0," EU DataGrid, Tech. Rep., 2003.

[12]. P. G., E. E., L. J, O. M., and T. S., "Scalable grid-wide capacity allocation with the swegrid accounting system (sgas)," Concurr.Comput. : Pract. Exper., vol. 20, pp. 2089–2122, Dec 2008.

[13]. A. Barmouta and R. Buyya, "Gridbank: A grid accounting services architecture (gasa) for distributed systems sharing and integration," in Proceedings of the 17th International Symposium on Parallel and Distributed Processing, ser. IPDPS '03. Washington, DC, USA: IEEE Computer Society, 2003, pp. 22–26.

[14]. G. von Voigt and W. Muller, "Comparison of grid accounting concepts for d-grid," in Proceedings of Cracow Grid Workshop 2006. ACC Cyfronet AGH, October 2006, pp. 459–466.

[15]. NexR, "icube cloud computing and elastic-storage services," Mar 2011. [Online]. Available: https://testbed.icubecloud.com

[16]. H. Rajan and M. Hosamani, "Tisa: Toward trustworthy services in a service-oriented architecture," IEEE Transactions on Services Computing, vol. 1, pp. 201–213, 2008.

[17]. S. Meng, L. Liu, and T. Wang, "State monitoring in cloud datacenters," IEEE Transactions on Knowledge and Data Engineering, vol. 23, pp. 1328–1344, 2011.

[18]. C. Olston and B. Reed, "Inspector gadget: a framework for custom monitoring and debugging of distributed dataflows," in Proc. of the 2011 international conference on Management of data, ser. SIGMOD '11. ACM, 2011, pp. 1221–1224.

[19]. P. Leitner, A. Michlmayr, F. Rosenberg, and S. Dustdar, "Monitoring, prediction and prevention of SLA violations in composite services," in Proc. of the 2010 IEEE ICWS. IEEE Computer Society, 2010, pp. 369–376.

[20]. S. Pearson and B. Balacheff, Trusted computing platforms: TCPA technology in context, ser. HP Professional Series. Prentice Hall PTR, 2003.

[21]. I. P. Release, "White paper: Trusted execution technology, hardware-based technology for enhancing server platform security," Intel Press, Tech. Rep., 2010.

[22]. A. Haeberlen, "A case for the accountable cloud," SIGOPS Oper. Syst. Rev., vol. 44, pp. 52–57, April 2010.

[23]. F. Koeppe and J. Schneider, "Do you get what you pay for? using proof-of-work functions to verify performance assertions in the cloud," in Cloud Computing Technology and Science (Cloud-Com), 2010 IEEE 2nd Intl. Conf. on, 2010, pp. 687–692.
[24]. R. Buyya, D. Abramson, J. Giddy, and H. Stockinger, "Economic models for resource management and scheduling in grid computing." Wiley Press, 2002, pp. 1507–1542.
[25]. B. N. Chun and D. E. Culler, "Market-based proportional resource sharing for clusters," Tech. Rep., 1999.