A Trustworthy and Computationally Efficient Cloud Service Contract System via Cloud Notary Authorities

Ki-Woong Park · Kyu Ho Park

Recording and keeping account of the usage of cloud resources in a trustworthy way are vital requirement for widespread cloud deployment and availability. To provide a trustworthiness of cloud service contracts, we identified that the frequent cryptographic operations for the trustworthy contracts lead to excessive computations and a bottleneck of contract transactions. As a remedy for these limitations, we propose a trustworthy contract system for cloud computing environment. The system, which introduces the concept of a Cloud Notary Authority (CNA) for the supervision of contract, makes contract more objective and acceptable to users and cloud service providers. CNA generates trustworthy binding information that can be used to resolve future disputes between a user and a cloud service provider. Because our contract system drastically minimizes the asymmetric key operations of cloud users and service providers, it provides a level of security that is identical to that of a Public Key Infrastructure (PKI) and it minimizes the latency of contract transactions. This work was undertaken on a real cloud computing service called iCubeCloud.

Keywords: Cloud service contract, Trustworthiness, Transaction processing, Pricing and resource allocation

I. INTRODUCTION

Cloud services change the economics of computing by enabling users to pay only for the capacity that they actually use. Service providers widely use the pay-per-use contract scheme in their pay-as-you-go pricing model: that is, the consumer uses as many resources as needed and is billed by the provider for the amount of resources consumed by the end of an agreed period. In this environment, the ability to record and account for the usage of cloud resources in a trustworthy way encourages widespread cloud deployment and availability [1],[23]. On the basis of trustworthiness, Cloud Service Providers (CSPs) and users can both construct trustworthy usage records to prove which resources were allocated and when they were initiated and released. And the usage record should describe all the factors that are taken into account in calculating resource consumption charges. In commercial cloud services, on the other hand, such as Amazon EC2, S3 [14] and Azure [3], the contract transactions and management are both processed by the CSP alone; there is no trustworthiness of contract transactions.

Among attempts to enforce the trustworthiness of contract systems, a Public Key Infrastructure (PKI)-based digital signature [17] stands out as a fundamental, widely-used mechanism. However, the computational complexity of a PKI may result in a high computational overhead and an intolerable contract response time because the asymmetric key operations of digital signatures must be performed for both the thin client terminal and the CSP. This problem is exacerbated when the number of contract transactions increases rapidly in proportion to the number of users that require diverse cloud resources.

By thoroughly investigating conventional contract systems of cloud computing environment, we identified

논문번호: TR11-030, 논문접수일자:2011.03.16, 논문수정일자:2011.04.04, 논문게재확정일자:2011.04.25
Ki-Woong Park, Kyu Ho Park: Korea Advanced Institute of Science and Technology

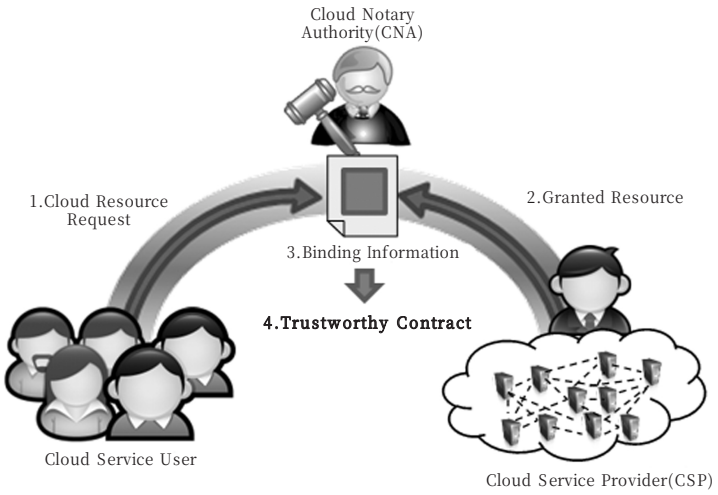


Figure 1. Cloud notary authority(CNA) for a trustworthy cloud service contract

the following two fundamental requirements, which drive the architecture of our contract system for cloud computing environemnt:

① Support for a trustworthy contract mechanism: In traditional cloud contract systems, the contract transactions and contract information are both processed by the CSP alone. For a trustworthy way of logging resource usage, a digital signature is essential because it enhances the contract mechanism with mutual trustworthiness [2]. As shown in Figure 1, our proposed contract system introduces the concept of a cloud notary authority (CNA) for the supervision of contract transactions; and it generates trustworthy binding information for users and CSPs. Furthermore, the resource usage log, which is based on a one-way hash chain, retains the information in its local storage for future accusations. CNA supervises the contract and, because of its objectivity, is likely to be accepted by users and CSPs alike.

② A computationally efficient contract mechanism with a minuscule computational overhead: The frequent cloud contract transactions lead to an excessive computational overhead or contract system bottleneck. To mitigate these problems, we propose a computationally efficient contract scheme which provides mutual trustworthiness. The proposed contract scheme drastically minimizes the asymmetric key operations of cloud users and CSPs; it provides a level of mutual trustworthiness that is identical to that of a PKI; and it minimizes contract transaction latency.

This paper is an extended work related to our previous work, called THEMIS [27]. Previous work mainly focuses on a protocol design to provide a mutual trustworthiness needed for a mutually verifiable billing system. In this paper, attempts are given to integrate a complete cloud contract system into resource management module of the cloud computing platform, called *iCubeCloud* [26] as the key primitive for a secure and trustworthy conatract transaction. Additionally, we present the performance results of our prototype with deeper insight to show its feasibility and practicality in terms of transaction latency, throughput, and overhead from the trustworthy contract mechanism.

The remainder of the paper is organized as follows: In Section II, we discuss relevant works. In Section III, we present the overall system design and components of the proposed contract system. In Section IV, we illustrate the proposed contract protocol and its trustworthness. In Section V, we describe our real implementation of the contract system on a real cloud computing service. In Section VI, we evaluate the performance of the proposed contract scheme. Finally, in Section VII, we present our conclusions and suggest future works.

II. PREVIOUS WORKS

A contract system that tracks the usage of computing re-sources has been actively studied and developed in the research area of grid or cloud computing. To date, how-ever, none of the contract systems have incorporated a

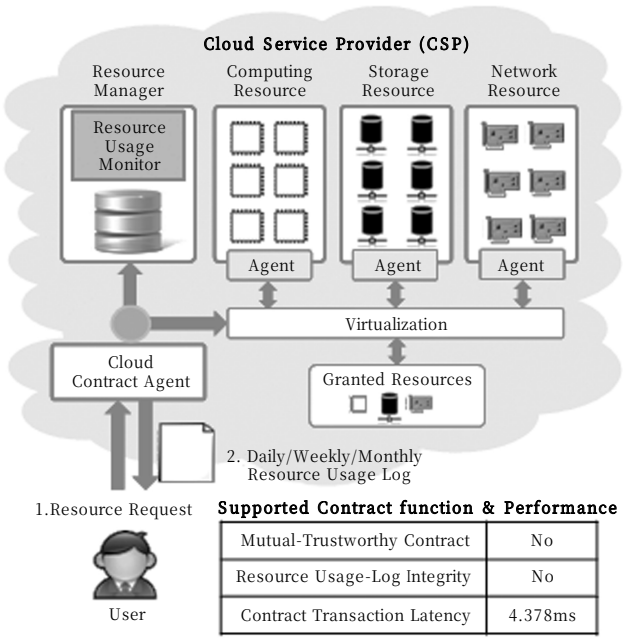


Figure 2. Native contract system and a brief overview of the characteristics in terms of contract function and transaction latency

trustworthy or computationally efficient contract mechanism. In this section, we briefly discuss the experimental results as we evaluate existing contract systems in terms of their security level and contract overhead. A more comprehensive evaluation of the experimental results is described in Section VI.

1. Native contract systems

In a pay-as-you-go pricing model, users can scale the capacity of cloud resources on demand. Resource consumption is billed on a utility basis with little or no upfront cost. Two pioneering studies identified challenges in managing the resources of a grid computing environment and proposed a computational economy as a metaphor for effective management of resources [4],[5]. Several researchers presented a resource usage processing system that can be used to scan batch system logs to build accounting records [6]~[10],[21]; this system is able to record and account for the usage of grid resources.

Figure 2 shows the architecture and characteristics of a contract system without any security concerns. The resource usage information, which pertains to the CPU cycles, storage, and network bandwidth, is collected via a resource usage monitor and charged over the cloud contract agent. APEL [6] presents a contract system that processes log information to create quantified accounting records. There are also other resource management and contract frameworks that have been suggested as part of traditional grid approaches: namely, Condor/G [7], Nimrod/G [8], GRASP [9], Tivoli [10], and TeraGrid [21]. However, rather than address security concerns, they focus on notions such as distributed resource usage metering and an accounting and account balancing mechanism for a distributed grid environment. Thus, they fail to provide the type of mutual trustworthiness and integrity needed in a trustworthy contract system that records the usage of cloud resources.

2. Security-enhanced contract systems

Several electronic payment schemes have been proposed in the literature in an attempt to provide security-enhanced contract mechanisms. They include micropayment-based schemes such as PayWord [18], MiniPay [16], e-coupons [19], and NetPay [24]. Broadly deployed in e-payment systems, these schemes enable users to securely and efficiently perform repeated payments. Many of these schemes are based on the use of one-way hash functions that generate chains of hash values; users perform contract transactions by releasing a certain number of hashes in the hash chain. Although these schemes are supposed to provide secure contract for micropayment transactions in a computationally efficient

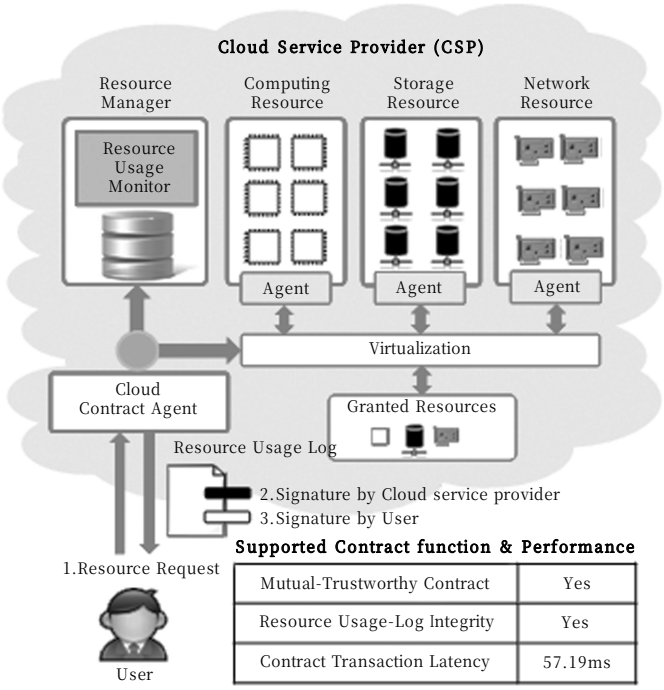


Figure 3. Contract system with PKI and its supported contract function and transaction latency

way, they offer no support for mutual trustworthiness of the resource usage log.

The research area of cloud and grid computing has presented the PKI-enhanced contract and accounting frameworks such as RUS [28], DGAS [11], SGAS [12], GridBank [13], and Amazon EC2 [14]. They have an access control mechanism that uses digitally signed certificates to define and enforce an access policy for a set of distributed resources. However, none of these projects treats the problem of dealing with the trustworthiness of contract transactions or its computational efficiency, as we do for the cloud computing environment.

Due to its digital signature and non-repudiation features, a PKI is generally considered to be the most appropriate and fundamental way of enforcing trustworthiness in terms of our requirements [15],[25]. Figure 3 illustrates the organization of a PKI-enhanced contract system and its characteristics in terms of security level and contract overhead. A PKI has a contract overhead even though it relies on full security features to achieve mutual trustworthiness and data integrity. The extent of the overhead is mainly determined by the extremely high complexity of asymmetric key operations when the PKI is used for a contract system by a thin client or heavily loaded server [20]. Figure 3 shows that the estimated average contract transaction latency in our experimental environment is 57.19 ms.

III. DESIGN OF THE CLOUD CONTRACT SYSTEM

1. Design philosophy

In this section, we present the overall architecture and contract process of the cloud contract system. While deliberating on the system requirements mentioned above, we based the design of our contract system and protocol on two principles:

- Trustworthy contract with a CNA: We devised the CNA to ensure that cloud resource usage is recorded and accounted for in a trustworthy and credible way. The CNA supervises contract transactions by generating binding information between a user and resource usage log and retains the information in its local storage for future accusation. Our designed CNA, which covers about 20,000 cloud service users, is based on a model of the computer usage pattern of users [22]. Currently, we are investigating the CNA from the perspectives of

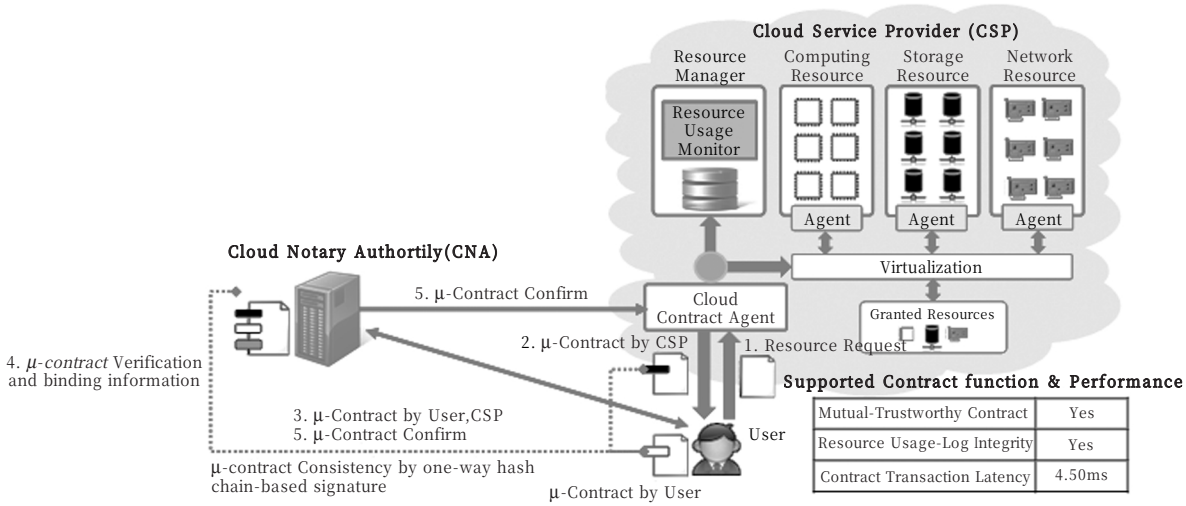


Figure 4. Flowchart of the overall architecture and contract transactions of the proposed cloud contract system

massive scalability and robustness. We believe that putting multiple trusted third parties in charge of the CNA is an appropriate way forward, as is the case with the PKI.

- Contract transactions with a minuscule computing overhead: The requirement of performing frequent transactions of frequent, trustworthy contract leads to an excessive computation overhead or produces a bottleneck in the contract system. To provide a non-obstructive contract operation, we propose a computationally efficient contract protocol that provides mutual trustworthiness and integrity of the resource usage log. The proposed contract protocol drastically minimizes the asymmetric key operations of cloud users and CSPs; it also minimizes the contract transactions latency and provides a level of mutual trustworthiness that is identical to that of a PKI.

2. The proposed cloud contract system

Figure 4 shows the overall architecture of our cloud contract system. The three major components of the architecture are listed as follows:

- Cloud Service Provider (CSP): The CSP enables users to scale their capacity upwards or downwards in accordance with their computing requirements and to pay only for the capacity that they actually use.
- Users: We assume that users are thin clients who use services in the cloud computing environment. To use services in such an environment, each user makes a resource request to the CSP with a contract transaction.
- Cloud Notary Authority (CNA): The CNA provides a trustworthy integrity mechanism that combats the malicious behavior of users or the CSP. In addition, it investigates contract transactions and generates trustworthy binding information among all the involved entities on the basis of a one-way hash chain and resource usage log; it also retains the information in its local storage for future accusations. The CNA enables our proposed contract system to provide mutual trustworthiness and integrity for a cloud resource usage log. The process, which involves exclusive key sharing between entities and a one-way hash chain-based signature, is computationally efficient for a thin client and the CSP.

3. Overall process of contract transaction

With the proposed CNA, trustworthy contract can be provided without asymmetric key operations of any entities after a registration phase. Figure 4 shows the overall process of a contract transaction with our contract system. The following details of the trustworthy contract mechanism are described in more detail in Section IV:

① The user generates a cloud resource request message and sends it to the CSP.
② The CSP sends the user a *μ-contract-CSP* generated with a digital signature from a CSP hash chain.
③ The user generates a *μ-contract* with a hash chain-based digital signature of the user and sends it to the CNA.

④ The CNA performs transactions to verify the *μ-contract* from the user, and generates trustworthy binding information between the user and the CSP for consistency of the *μ-contract*.

⑤ The contract process is completed when the user and the CSP receive confirmation from the CNA.

IV. CONTRACT PROTOCOL AND ITS TRUSTWORTHNESS

While deliberating on our system design philosophy, we diligently tried to streamline the computation overhead of the contract operation. Our novel *μ-contract* is generated by a hash function and exclusively distributed keys among entities. It optimizes the computation overhead of the contract mechanism and facilitates mutual trustworthiness and integrity for cloud resource usage.

1. Operational Flow of the contract protocol

In this section, we describe an operational flow of the overall transactions of the proposed contract protocol. More specific data structure of each message is described in [27]. The protocol consists of the following three states:

- State 1 (Mutul Authentication): When a user first accesses the CSP, mutual PKI-based authentications are performed by the user, the CSP, and the CNA. Throughout the mutual authentications, the user, the CNA, and the CSP exclusively share the following three keys:

$CSP \leftrightarrow CNA:(K_{c,n})$, $User \leftrightarrow CNA:(K_{u,n})$, and $User \leftrightarrow CSP:(K_{u,c})$

- State 2 (Hash Chain Registration): In this state, the user, the CSP, and the CNA generate a hash chain of length N by applying the hash function *N* times to a seed value ($C_{u,N}$, $C_{c,N}$, and $C_{n,N}$) so that a final hash ($C_{u,0}$, $C_{c,0}$, and $C_{n,0}$) can be obtained. The user and the CSP commit to the final hash by digitally signing the final hash ($C_{u,0}$ and $C_{c,0}$), and by registering the digital signed hash elements to the CNA. The purpose of this registration is to commit the final hash values to the CNA and to receive the final hash ($C_{n,0}$) generated by the CNA. Once the commitment of the one-way hash chains is successfully completed, *State 2* is skipped until the corresponding hash chain either expires or is revoked.

- State 3 (Contract Transaction): A user who intends to receive a cloud resource from the *CSP* generates a resource request message encrypted with $K_{u,c}$ and sends it to the *CSP*. Upon receiving the message, the *CSP* transmits a stipulation (S) containing an agreement that covers factors such as the granted resource, the time, and the price as well as the *μ-contract-CSP*. The *μ-contract-CSP* contains an encrypted value of three inputs (namely, a hashed value of the stipulation (*S*), a hash element ($C_{c,n}$), and a randomly generated nonce) and a hashed value of two inputs (namely, a stipulation (*S*) and a hash element ($C_{c,n}$)). The hash element is updated for each contract transaction on a chain-by-chain basis so that all of the hash elements can be linked and verified sequentially toward the seed value ($C_{u,0}$, $C_{c,0}$, and $C_{n,0}$) of the hash chains. Furthermore, the used hash element ($C_{c,n}$) is unknown to the user. After receiving the *μ-contract-CSP*, the user generates a notary request message by combining the *μ-contract-CSP* from the CSP with the user's own *μ-contract-User*. In this message, the stipulation (*S*) used to generate the *μ-contract* is unknown to the CNA. The user then sends the notary request message to the CNA for the mutual trustworthness and integrity of the *μ-contract*. When the message arrives, the CNA compares the H(S) section of μ-contract-User with the *H(S)* section of *μ-contract-CSP* to check the justness of the contract request message. If the two contexts are identical to each other, the CNA sends the user and CSP a confirmation message, which contains the hashed value of *H(S)* and the hash element ($C_{n,n}$) of the CNA. Otherwise, the CNA sends an error message to the user and CSP. The contract transaction is completed when the user and the CSP receive the final confirmation message. The verified *μ-contract* is subsequently retained on the local repository of the CNA for future accusations.

The proposed protocol can provide a trustworthy and non-obstructive method of contract after the authentication (*State1*) and the hash chain registration (*State2*). Any user who accesses the CSP for the first time or needs the hash chain renewal is asked to perform a mutual authentication (*State1*) or generate a hash chain and commit to it by digitally signing the final hash (*State2*), which requires an asymmetric key operation. From the perspective of computation efficiency, the initial contract operation takes longer than a normal PKI-based contract operation. On the other hand, the contract overhead for the client and the

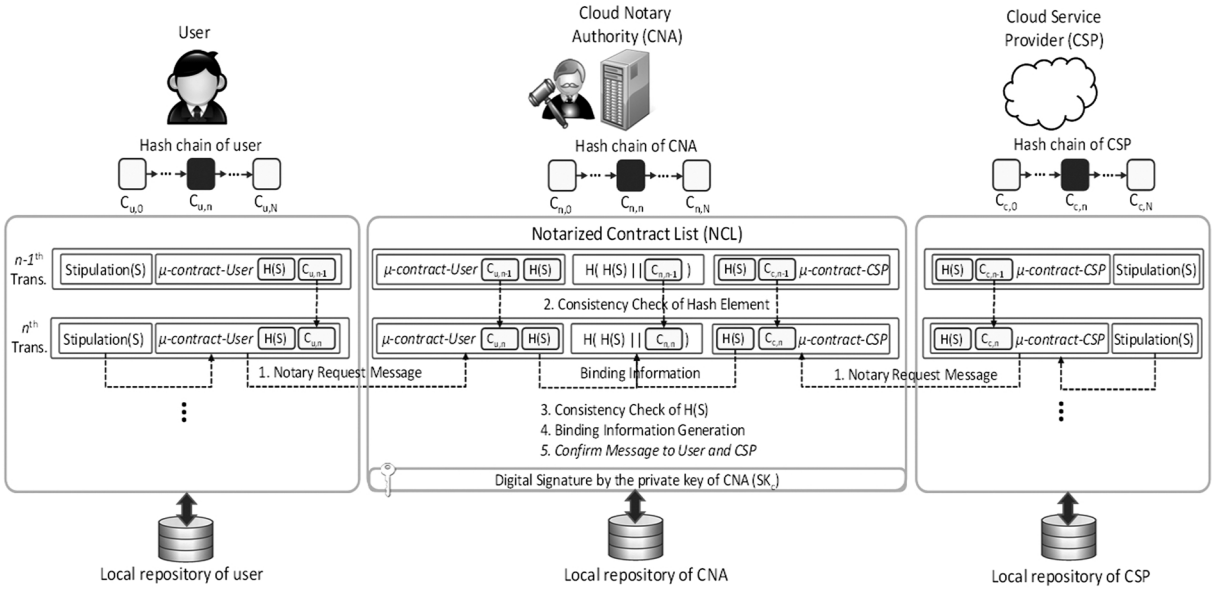


Figure 5. Overall message transactions and local repositories for mutually verifiable contract transactions

CSP can be reduced drastically after the completion of *State 2* because the user and the CSP can perform the contract transaction by simply using symmetric key and hash operations.

3. How the contract records can be trustworthy

Our proposed protocol can provide mutual trustworthiness and integrity through the *μ-contract*, which is generated among the entities by symmetric and hash cryptography and distributed keys ($K_{u,c}$, $K_{u,n}$, and $K_{c,n}$). This section elaborates how the contract can be verified in collaboration with the CNA. Trustworthy contract can be achieved when the CNA verifies the validity of the user's message for each contract transaction in *State 3*. As described in Figure 5, when the notary request message arrives at the CNA, the CNA examines the consistency of the hash elements ($C_{u,n}$ and $C_{c,n}$) from the user and the CSP by checking the link with the previously used hash elements ($C_{u,n-1}$ and $C_{c,n-1}$). Next, the CNA verifies the consistency between the *H(S)* section of *μ-contract-User* and the H(S) section of *μ-contract-CSP*. By the checking, the CNA can prove that the CSP sent the *μ-contract-CSP* and that the user sent the notary request message with the same stipulation (*S*) as the stipulation from the CSP. The CNA subsequently generates binding information, which contains the hashed value of *H(S)* and the hash element ($C_{n,n}$) of the CNA. The CNA then sends the user and CSP a confirmation message with the binding information. Upon receiving the message, the user and CSP confirm that the corresponding contract transaction (*μ-contract*) is correctly notarized with the hash element ($C_{n,n}$) of the CNA. After each contract transaction, the CNA retains the corresponding *μ-contract*, binding information, and $C_{n,n}$ at their local repository; the user stores the contract-related information (*S* and *μ-contract-User*); and the CSP stores the information (*S* and *μ-contract-CSP*) as evidentiary data. In the case of the CNA, the transactional data is stored in a type of notarized contract list (NCL). The NCL is a XML-based data structure for storing evidence of the contract transactions for future accusations. All of the contexts are periodically stored with the digital signature of the CNA to ensure the integrity of the NCL context.

Figure 6 illustrates how the NCL is used to prove the integrity of certain contract transactions with the verification module of the CNA. The verification module consists of three hash modules: the *User-Verifier*, the *CNA-Verifier*, and the *CSP-Verifier*. The *CNA-Verifier* verifies the integrity of the stipulation (S) from the user or CSP by comparing the stipulation with the binding information of the CNA. The *User-Verifier* and the *CSP-Verifier* check the correctness of an asserted contract transaction by the user and CSP, respectively. Each verifier inputs a stipulation (S) from the user or CSP as

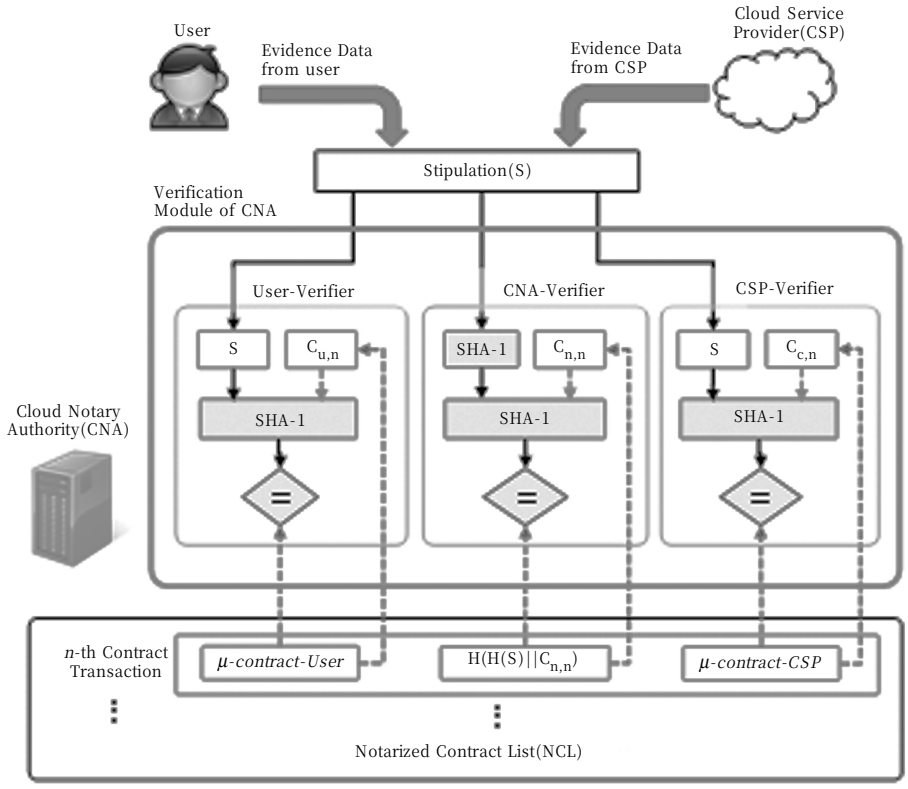


Figure 6. Verification mechanism with the notarized contract list (NCL) in the CNA

well as hash elements from the NCL to verify the integrity of certain contract transactions. For example, in the case of the *User-Verifier*, a discrepancy between the output of the hash function and the stored data of the CNA proves that the user has modified the stipulation of the relevant contract.

Whenever the CSP asserts that a user repudiates a certain contract transactions, the CSP can submit a claim for justice to the CNA, drawing attention to the stipulation (*S*) included in the corresponding *μ-contract-CSP*. The CNA then verifies the claim by comparing the output value of the *CNA-verifier* with the corresponding binding information stored in the NCL. If the claim is correct, the CNA then demands to see the stipulation (*S*) used to generate the *μ-contract-User*. The CNA then inputs the stipulation (*S*) from the user and the hashed value stored on NCL into *User-Verifier* and inputs the stipulation (*S*) from CSP and the hashed value stored on the NCL into the *User-Verifier*. Any discrepancy between the output of the hash function and the stored data of the NCL proves that the user or CSP has modified the stipulation of the relevant contract.

V. SYSTEM IMPLEMENTATION

In this section, we describe our real implementation of the cloud contract system on a real cloud computing service called *iCubeCloud* [26]. First, we explain the purpose and features of the three key components in detail. We then describe the overall system interface with the process for passing messages among the three entities of the cloud contract system.

1. Overall software component

Figure 7 shows the overall software components of the cloud contract system. The three major components are the Contract-Client, the Contract-CSP, and the Contract-CNA. The components interact with each other on the basis of the proposed contract protocol via a secure communication. For secure communication and cryptographic operations, we deployed OpenSSL as cryptography primitive. The performed contract transactions are maintained by using MySQL and SQLite in the Contract-Client, the Contract-CSP, and the

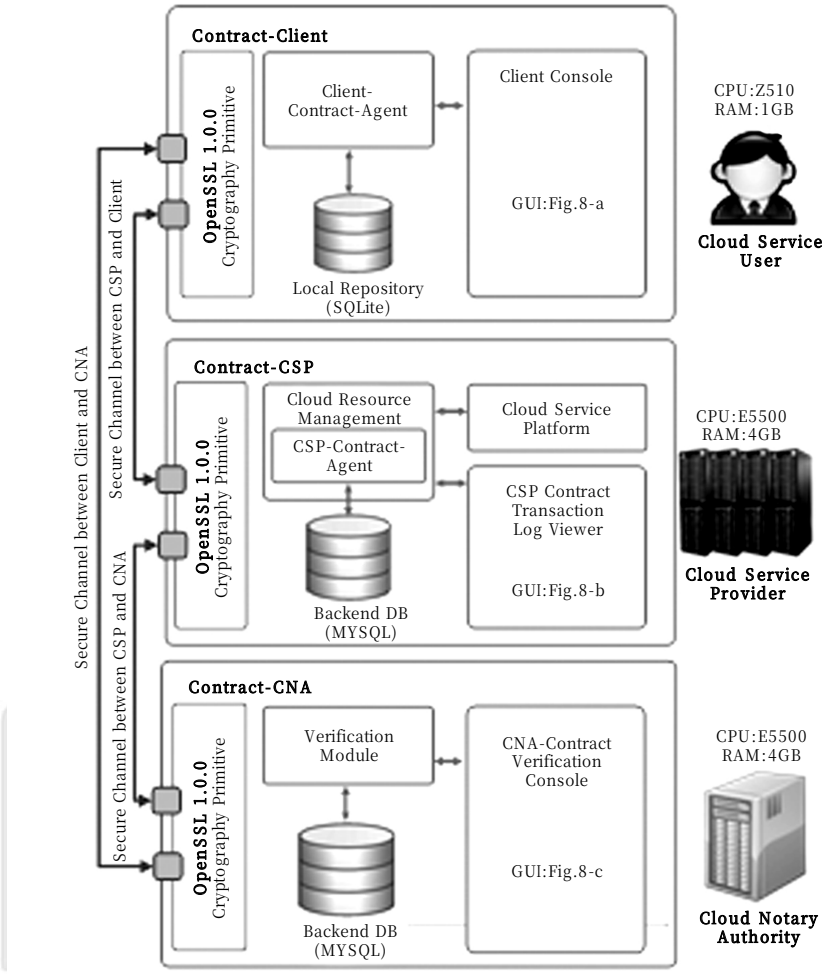


Figure7. Overall software architecture of the cloud contract system

Contract-CNA. We integrated our implemented system in two kinds of machines with the different hardware specifications. A client running the Contract-Client has an *Intel Z510* processor and a 1GB main memory; the other components have a *Xeon E5500* processor and 4 GB main memory. The three major components of the overall contract system are described as follows:

- Contract-Client: The Cloud-Client is a client side application that provides the user interface for working with the CSP and the CNA. Users can start and stop user-defined virtual machine instances, view and perform actions on running instances, and manage storage volumes with simple clicks of a mouse. They can also check the consumption of cloud resources such as the CPU, memory, and storage capacity. The core part of the Contract-Client is the client contract agent module. When users make a resource request message, the proposed contract protocol is performed on the client contract agent. The transactional data of the protocol are then stored into the local repository.
- Contract-CSP: The Contract-CSP acts as a mediator between users and the cloud service platform. The core part of the Contract-CSP is the CSP contract agent module. The CSP contract agent is a part of our underlying cloud computing platform, called as *iCubeCloud* [26]. In our platform, the CSP contract agent is integrated into the resource management module as the key primitive for a secure and trustworthy contract transaction. When a user who intends to receive a cloud resource sends a resource request message, the CSP contract agent performs the contract transaction on the basis of the proposed protocol by generating a *μ-contract-CSP* and assigning

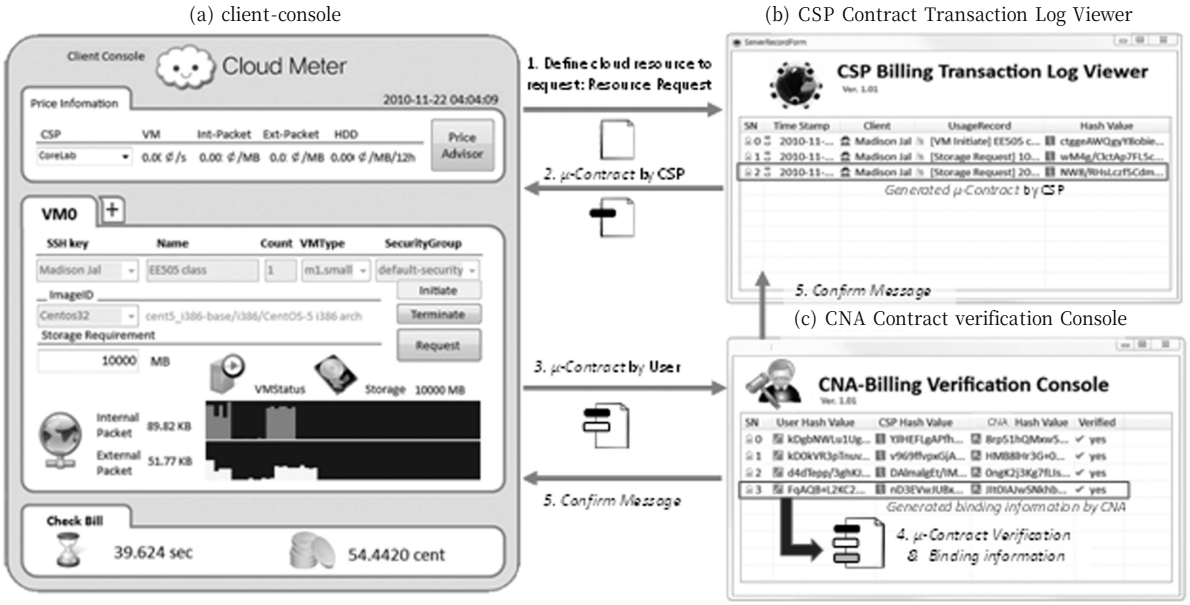


Figure 8. Screenshot of (a) client console (b) log viewer of CSP, and (c) verification console of CNA

the requested cloud resources that interact with the cloud resource management module. After that, the CSP Contract agent records the contract transactions in the backend database for future accusations. The performed contract transactions can be viewed via the CSP contract transaction log viewer.

- Contract-CNA: The Contract-CNA consists of two submodules: a verification module and a backend database. The verification module supervises the contract and makes the contract more objective and acceptable to users and CSPs. After receiving a notary request message, the verification module verifies the received μ-contract, generates the binding information, and records that information in the backend database for future accusations. Through this process, the CNA provides mutual trustworthiness and integrity for a cloud resource usage record, affording protection against the malicious behavior of users or the CSP. The CNA-contract verification console can be used to verify a certain contract transaction at the request of the user or the CSP. The console easily demonstrates what happens if a user or CSP engages in malicious behavior.

2. Overall system interface

The screenshot of the above-mentioned three entities in Figure 8 illustrates one unit of the contract transaction. The screenshot is divided into three parts. The first part (Figure 8-a) is the client-console interface where users can start and stop user-defined virtual machine instances, view and perform actions on running instances, and manage storage volumes. The second part (Figure 8-b) is the CSP contract transaction log viewer, which displays the performed contract transactions; it covers factors such as the granted resource, the time, the price, and the *μ-contract-CSP*, which is implemented inside the CSP. The third part (Figure 8-c) is the CNA contract verification console; it displays the notarized contract transactions implemented inside the CNA. The whole process is described as follows:

① At the client console on the user side, the user can request a cloud computing resource by selecting a virtual machine type, a system image, the storage capacity, and so on.
② After receiving the request message from the user side, the contract agent of the CSP sends the user a *μ-contract-CSP* message corresponding to the resource request message.
③ After acquiring the message from the contract agent of the CSP, the client console program generates a *μ-contract-User* and sends it to the CNA.
④ The CNA performs transactions to verify the *μ-contract* from the user and generates trustworthy binding information between the user and the CSP for consistency of the *μ-contract*. The *μ-contract* and the

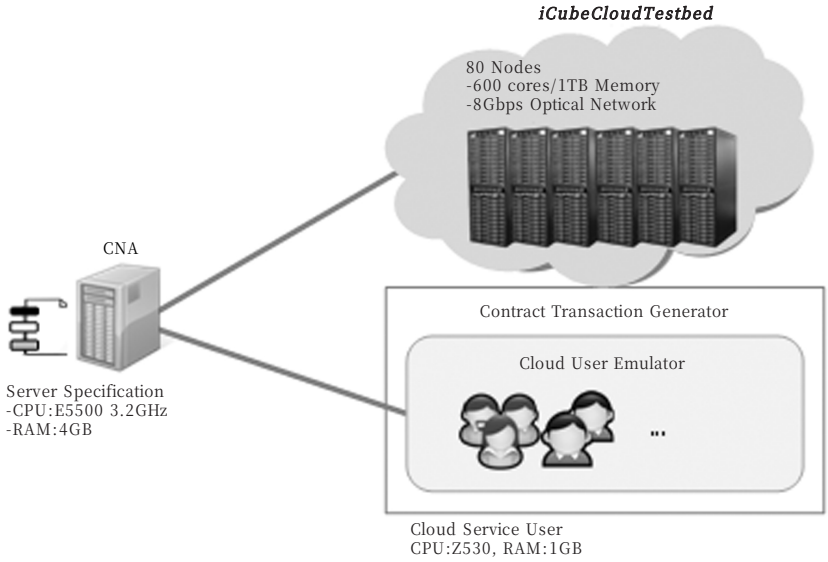


Figure 9. Experiment environment for measuring the performance of the cloud contract system in terms of the contract transaction latency and operational efficiency

corresponding binding information are then stored in the backend database.

⑤ The contract process is completed when the client console and the contract agent of the CSP receive confirmation from the CNA.

Through this process, one unit of the contract transaction is accomplished. All transactional data are stored in their own database, which can be displayed in each logger of the components.

VI. EFFICIENCY AND PERFORMANCE EVALUATION

In this section, we present the performance results of our prototype version of the cloud contract system. First, we demonstrate the overall experimental environment. We then describe the operational efficiency of the contract protocol to evaluate the performance of the cloud contract system in terms of latency and throughput of the proposed contract protocol.

1. Experimental environment

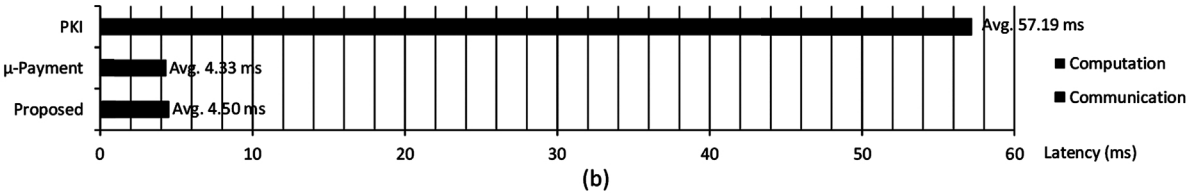
To evaluate the performance characteristics of the cloud contract system, we constructed a cloud user emulator and coupled it to a contract transaction generator. The objective of the emulator is to simulate the processing and communications resources anticipated in a full implementation. Figure 9 shows the overall experimental environment. The operating times of the emulator are similar to actual operating times (for communications, contract operations, and message processing). In addition, the user emulator receives control signals from the contract transaction generator. The generator is a module that generates control signals to produce contract request messages; for this purpose, we use a random generator that models the computer usage pattern of users [22].

2. Cryptography operation experiment

The first experiment measures the cryptography processing time on the client side, which is equipped with an *Intel Z510*, 1GB RAM and a server (an *E5500* processor with a 4 GB RAM). Table 1 compares the processing time of an RSA 1,024 bit algorithm [20] as an asymmetric key operation, an AES 128 bit algorithm as a symmetric key operation, and a SHA-1 algorithm as a hash function. It shows that the times required to decrypt and encrypt a 128 byte block of data with the RSA 1,024 bit algorithm are 22.324 ms and 1.169 ms, respectively, on the computing devices of the client. In contrast, only 0.014 ms is required to encrypt and decrypt the block with the AES 128 bit algorithm on the device because an AES operation has a much lower computation overhead than

System	Security Functionality		User				Cloud Service Provider (CSP)				Notary Authority (Trust 3rd Party)				Communication	Total Transaction Time (Computation + Communication)
	Mutual Verifiability	Integrity	RSA		AES	SHA	RSA		AES	SHA	RSA		AES	SHA		
			PU	PR			PU	PR			PU	PR				
Native PKI-based Contract	Yes	Yes	2	1	3	0	1	2	3	0	1	0	0	0	Avg.13.81ms	Avg.57.19ms
			Avg. 36.611 ms				Avg. 5.203 ms				Avg. 1.567 ms					
μ-Payment	No	No	0	0	4	0	0	0	0	0	0	0	4	1	Avg. 4.221ms	Avg. 4.33ms
			Avg. 0.084 ms				Avg. 0.0 ms				Avg. 0.019 ms					
Proposed	Yes	Yes	0	0	4	2	0	0	3	1	0	0	4	2	Avg. 4.379ms	Avg. 4.50ms
			Avg. 0.087 ms				Avg. 0.016 ms				Avg. 0.019 ms					

(a)



(b)

Figure10. (a) the number of public/private keys and the symmetric key operations with the total operation time for each protocol(PKI, μ-payment,proposed) (b) contract transactions latency

Table1. Processing times of encryption,decryption,and hash for each algorithm and operation environment

Platform	Cryptography		Operation time
• Client Side - CPU:Intel Z510 - RAM:1GB	RSA 1024bit	Private Key	Avg. 22.324 ms
		Public Key	Avg. 1.169 ms
	AES 128bit	Encryption	Avg. 0.014 ms
		Decryption	Avg. 0.014 ms
	Hash Function	SHA-1	Avg. 0.0012 ms
• Server - CPU:Xeon E5500 - RAM:4GB	RSA 1024bit	Private Key	Avg. 2.012 ms
		Public Key	Avg. 0.117 ms
	AES 128bit	Encryption	Avg. 0.0031 ms
		Decryption	Avg. 0.0031 ms
	Hash Function	SHA-1	Avg. 0.0003 ms

the RSA algorithm. Furthermore, SHA-1 needs only 0.0012 ms of operation time to generate a hashed message. On the server side, the cryptography operation time is drastically reduced by a high-performance processor and a huge memory. On the server side, a 128 byte block of data with the RSA 1,024 bit algorithm is decrypted in 2.012 ms and encrypted in 0.117 ms. The same block with the AES 128 bit algorithm on the device is encrypted and decrypted in 0.0031 ms. The SHA-1 takes only 0.0003 ms of operation time to generate a hashed message.

3. Contract protocol efficiency and comparative evaluation

The performance of the contract protocol in terms of the contract overhead and the consumption of processing and communication resources is an important factor to be considered when designing contract protocols. First, we analyze how the efficiency of the proposed contract system compares with PKI-based contract; we also analyze the μ-payment in terms of computation and communication efficiency.

Figure 10 (a) shows the number of public and private keys (RSA 1,024 bits), the symmetric key (AES 128 bits), and the hash (SHA-1) operations performed with the total operating time for each contract protocol. In spite of its smaller number of cryptography operations per contract, the PKI-based contract protocol has a much longer latency period for contract transactions than other schemes because it has a certain number of private and public key

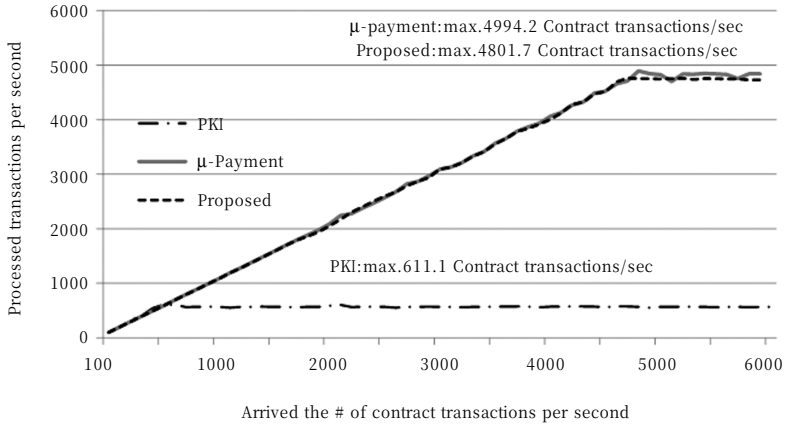


Figure 11. Throughput of the contract transactions with varying numbers of contract transactions per second

operations for all of the entities. The μ-payment, on the other hand, has the shortest processing time because it can be completed by symmetric key and hash operations, but it fails to meet our security requirement. In the case of the proposed contract system, the first time a user accesses the CSP, all of the entities perform mutual PKI-based authentications and generate a hash chain which requires two private and public key operations and multiple hash operations. The authentication and hash chain generation time of the proposed contract system is similar to the operating time of PKI-based contract. However, after the operations, the user can perform a contract operation by processing only four symmetric key operations and two hash operations. This process has a much shorter operating time than the corresponding process of a PKI-based contract transaction. By way of summarizing the above results, we give an outline of the overall transaction time of the contract protocols. As shown in Figure 10(b), we estimate the operating time of each entity for each contract transaction so that we can measure how much the cryptography contributes to the contract overhead. The PKI-based contract transaction time with the RSA 1,024 bits algorithm is 57.19 ms. In the case of the μ-payment, the contract transaction can be accomplished without asymmetric key operations. These results confirm that even though the μ-payment has much shorter contract latency (4.33 ms), it cannot provide mutual trustworthiness and integrity. In the case of the proposed contract system, the contract transactions are accomplished without asymmetric key operations but still ensure mutual trustworthiness and integrity. We measured the contract transaction latency by measuring the interval between the resource request message and the confirmation message on the client side. The results confirm that, in the case of the proposed contract system, the total transaction time for the user and the CSP is much shorter than that of the PKI-based contract system. Moreover, without compromising the PKI security level, the total contract transaction time of the proposed contract system (4.50 ms) is much shorter than that of PKI-based contract (57.19 ms).

4. Throughput evaluation

Figure 11 illustrates how the throughput of the contract transactions mutates as the number of contract requests per second varies. The number of contract requests per second ranges from 100 to 6,000. For the PKI-based contract protocol, we found that the throughput is saturated on 611.1 transactions per second as the number of contract requests increases, mainly as a result of the cryptography operations and the communication overhead on both the client side and the server side. In the case of the proposed contract system and the μ-payment, the throughput is saturated on 4801.7 and 4994.2 transactions per second, respectively, as the number of contract requests increases. This phenomenon is due to the fact that quantity of user and server provider-side operations and the μ-payment is much smaller than that of PKI-based contract. The μ-payment cannot provide mutual trustworthiness and integrity; our contract system can provide them. This result confirms that the proposed contract protocol can seamlessly achieve fine-grained trustworthiness whenever the number of requests per second is less than 4,800.

5. Storage overhead

The CNA needs to store all the *μ-contracts* contained in the messages and binding information so that mutual trustworthiness can be ensured for an indefinite period. First, we measured the size of the data stored in the local repository of the CNA, the users, and the CSP for a period of one month. The *iCubeCloud* service is the CSP in this study; it is currently being used by five research institutes for course work and research activities. The total number of users is about 2,700. The average size of the *μ-contracts* stored by the CNA per user per day is 8.7?KB; the average size of the usage log and related information stored by the cloud and the user is 71.6?KB per user per day. If a million people use the cloud service, the cloud contract system will need archiving capabilities so that after a certain period records can be moved from the CNA into archival storage. Currently, we are working towards the scalability and optimization of the storage requirements for cloud contract transactions.

VII. CONCLUSION

Our aim was to provide a full-fledged trustworthy and non-obstructive contract solution tailored for a cloud computing environment. To accomplish this task, we thoroughly reviewed the ways in which conventional contract systems are used in the environment, and we consequently derived blueprints for our trustworthy and computationally efficient contract system. Besides utilizing conventional contract systems, we conceived and implemented the concept of a CNA that supervises contract to make it more objective and acceptable to users and CSPs alike. Our contract system features two remarkable achievements:

① Contract transactions with a minuscule computing overhead: Our trustworthy contract protocol replaces prohibitively expensive PKI operations without compromising the security level of the PKI; as a result, it significantly reduces the contract transaction overhead.

② Cloud Notary Authority (CNA): This entity ensures undeniable verification of any transaction between a cloud service user and a CSP. Our cloud contract system consequently represents a cost-effective but uncompromisingly secure development of a contract system. According to the performance evaluation, the contract overhead of our contract system (which averages 4.50 ms) is much shorter than the contract overhead of conventional PKI-based contract (which averages 57.19 ms). Thus, the throughput of the contract transactions of the proposed cloud contract system (4801.7 transactions per second) is much higher than the throughput of PKI-based contract transactions (which averages 611.1 transactions per second).

Our next step is to consider the scalability and fault tolerance of the cloud contract system. Currently, we are investigating the system from the perspectives of massive scalability and robustness. We believe that putting multiple trusted third parties in charge of the CNA is an appropriate way forward, as is the case with the PKI. We are working towards a coud contract system with more fault tolerance to scalable contract.

[References]

[1] Ristenpart T., Tromer E., Shacham H, and Savage S., "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds," 16th ACM Conference on Computer and Communication Security (CCS) 2009.

[2] H.-Y. Lin, and L. Ham, "Authentication protocols with non-repudiation services in personal communication system," IEEE Communications Letters 3 (8). 1999.

[3] Azure Service, "Microsoft, Windows Azure Platform," http://www.microsoft.com/windowsazure/

[4] Buyya R, Abramson D, Giddy J, Stockinger H. "Economic models for resource management and scheduling in Grid computing," Concurrency and Computation: Practice and Experience 2002; 14(13-15):1507-1542.

[5] Chun BN, Culler DE. Market-based proportional resource sharing for clusters. Technical Report CSD-1092, Computer Science Division, UC at Berkeley, Jan. 2000.

[6] R Byrom , "APEL: An implementation of Grid accounting using R-GMA, et al.," 2005 UK e-Science All Hands Meeting, Sep. 2005.

[7] M. Litzkow, M. Livny, and M. Mutka, "Condor: A hunter of idle workstations," 8th Int. Conf. Distributed Computing Systems (ICDCS 1988), San Jose, CA, Jan. 1988.

[8] Buyya R, Abramson D, Giddy J., "Nimrod/G: An architecture for a resource management and scheduling system in a global Computational Grid," Proc. of the 4th International Conference on HPC,

May 2000.
[9] Kwon, O., Hahm, J., Kim, S., and Lee, J. 2004., "GRASP: A Grid Resource Allocation System based on OGSA," In Proc. of the 13th IEEE international Symposium on High Performance Distributed Computing.
[10] IBM, "Tivoli: Usage and Accounting Manager," IBM Press 2009.
[11] Guarise, A., Piro, R., and Werbrouck, A. "DataGrid Accounting System - Architecture," EU DataGrid, 2003.
[12] P. Gardfjäll, E. Elmroth, L. Johnsson, O. Mulmo, and T. Sandholm. Scalable grid-wide capacity allocation with the SweGrid Accounting System (SGAS). Concurrency Computat.: Pract. Exper., 20(18):2089-2122, 2008.
[13] A. Barmouta and R. Buyya, "GridBank: A Grid Accounting Services Architecture (GASA) for Distributed Systems Sharing and Integration," Proceedings of the 17th IEEE IPDPS 2003, Apr.,2003, pp. 22-26.
[14] Amazon Web Services, "Amazon Elastic Compute Cloud (EC2) and Simple Storage Service (S3)," http://aws.amazon.com/ec2/, http://aws.amazon.com/s3/
[15] Thompson, M., Johnston, W., Mudumbai, S., Hoo, G., Jackson, K., and Essiari, "Certificate-based access control for widely distributed resources." In Proceedings of the 8th Conference on USENIX Security Symposium
[16] A. Herzberg, H. Yochai, "Mini-pay: charging per click on the web," in: 6th World Wide Web Conference, Santa Clara, USA, Apr. 1997.
[17] Stefan Kelm, "Public Key Infrastructure," http://www.pki-page.info/, 2007
[18] R. Rivest, A. Shamir, "PayWord and MicroMint: two simple micropayment schemes," 1996 International Workshop on Security Protocols, Lecture Notes in Computer Science, Vol. 1189, Springer, pp. 69-87
[19] V. Patil, R.K. Shyamasundar, "An efficient, secure and delegable micro-payment system," 2004 IEEE International Conference on e-Technology, e-Commerce and e-Service, Taipei, Taiwan, 28-31 Mar. 2004.
[20] J. Jonsson and B. Kaliski, "Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1," Network Group
[21] E. Roberts, M. Dahan, J. Boisseau. "TeraGrid User Portal: An Integrated Interface for TeraGrid User Information and Services," TeraGrid 2008.
[22] D.Banks, J.S.Erickson, and M.Rhodes, "Towards Cloud-based Collaboration Services," Usenix Workshop HotCloud 2009.
[23] N.Santos, K.P.Gummadi, and R.Rodrigues, "Towards Trusted Cloud Computing," Usenix Workshop HotCloud 2009.
[24] Dai, X., and Grundy, J. "NetPay: an off-line, decentralized micro-payment system for thin-client applications", E-Commerce Research and Applications, 6, 2007, 91-101.
[25] Park, K., Lim, S. S., and Park, K. H., "Computationally Efficient PKI-Based Single Sign-On Protocol, PKASSO for Mobile Devices", IEEE Trans. Computers. 57, 6 (Jun. 2008), 821-834.
[26] NexR. Co. Ltd., "iCube Cloud Computing and Elastic-Storage Service," http://testbed.icubecloud.com, 2010.
[27] Park, K., Park,S, Han, J; Park, K.H, "THEMIS: Towards Mutually Verifiable Billing Transactions in Cloud Computing Environment," Proc. of the 3rd IEEE Intl. Conf. on CLOUD , vol., no., 5-10 Jul. 2010, pp.139-147.
[28] J. Ainsworth, S. Newhouse, and J. MacLaren, "Resource Usage Service (RUS) based on WS-I Basic Profile 1.0 "Draft specification: draft-ggfwsi-rus-17, Aug. 2006.

Ki-Woong Park

Ki-Woong Park received the BS degree in computer science from Yonsei University in 2005 and the MS degree in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST) in 2007. He is currently working toward the PhD degree in the Division of Electrical Engineering at KAIST. His research interests include security protocol, network security, and cloud computing systems. He received a 2008 · 2009 Microsoft Graduate Research Fellowship. He is a member of the IEEE and the IEEE Computer Society.
E-mail: woongbak@core.kaist.ac.kr

Kyu Ho Park

Kyu Ho Park received the BS degree in electronics engineering from Seoul National University in 1973, the MS degree in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST) in 1975, and the DrIng degree in electrical engineering from the University of Paris XI in 1983. Since 1983, he has been a professor in the Division of Electrical Engineering and Computer Science at KAIST. From 2005 to 2006, he was the president of the Korea Institute of Next-Generation Computing. His research interests include computer architectures, file systems, storage systems, ubiquitous computing, and parallel processing. He is a member of KISS, KITE, the Korea Institute of Next-Generation Computing, the IEEE, and the ACM.
E-mail: kpark@core.kaist.ac.kr