THEMIS: A Mutually Verifiable Billing System for the Cloud Computing Environment

Ki-Woong Park, *Member*, *IEEE*, Jaesun Han, *Member*, *IEEE*,
JaeWoong Chung, *Member*, *IEEE*, and Kyu Ho Park, *Member*, *IEEE*

Abstract—With the widespread adoption of cloud computing, the ability to record and account for the usage of cloud resources in a credible and verifiable way has become critical for cloud service providers and users alike. The success of such a billing system depends on several factors: The billing transactions must have integrity and nonrepudiation capabilities; the billing transactions must be nonobstructive and have a minimal computation cost; and the service level agreement (SLA) monitoring should be provided in a trusted manner. Existing billing systems are limited in terms of security capabilities or computational overhead. In this paper, we propose a secure and nonobstructive 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. Furthermore, to provide a forgery-resistive SLA monitoring mechanism, we devised a SLA monitoring module enhanced with a trusted platform module (TPM), called S-Mon. The performance evaluation confirms that the overall latency of THEMIS billing transactions (avg. 4.89 ms) is much shorter than the latency of public key infrastructure (PKI)-based billing transactions (avg. 82.51 ms), though THEMIS guarantees identical security features as a PKI. This work has been undertaken on a real cloud computing service called *iCubeCloud*.

Index Terms—Records, verification, transaction processing, pricing, and resource allocation

1 Introduction

Cloud computing is representative of an important transition and paradigm shift in service-oriented computing technology. Emerging cloud services, such as Amazon EC2, S3 [1], and Microsoft Azure [2], have become popular in recent years. Although cloud computing has its roots in grid computing and utility computing technologies, it differs significantly from those technologies in terms of its service model.

Cloud service providers (CSPs) generally use a pay-per-use billing 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-upon period. CSPs usually guarantee the quality of service (in terms of availability and performance) in the form of a service level agreement (SLA) [3]. An SLA is supported by clear metrics and regular performance monitoring. In this service model, users who use an infrastructure-as-a-service (IaaS) may wish to figure out the billed charges for the total service time and the guaranteed service level. If a company uses a platform-as-a-service (PaaS) or software-as-a-service (SaaS), the accounting department of the company may require the service usage logs so as to verify the billed charges by checking the company's total number of running software programs or platforms. We refer to this type of transaction as a billing transaction; it is used to keep track of cloud service usage records and to verify whether the CSP has offered the quality of service promised under the SLA arrangement.

Providing a billing mechanism in a trusted manner is critical for CSPs and users [4]. However, the security aspects of a cloud billing system and the scale of cloud services often raise the following security and system issues:

- *A billing transaction with integrity and nonrepudiation capabilities*. For transparent billing of the cloud services, each billing transaction should be protected against forgery and false modifications [5]. Although commercial CSPs [1], [2] provide users with service billing records and while several researchers have presented resource usage processing systems [6], [7], [8], [9] that record the use of grid resources, they cannot provide a trustworthy audit trail. It is because the user or the CSP can modify the billing records even after a mutual agreement between the user and the CSP, leading to the dispute between them. In this case, even a third party cannot confirm that the user's record is correct or that the CSP's record is correct. Therefore, a trustworthy audit trail is important for resolving disputes, and the billing record in the billing transaction must be assuredly incorruptible per

- *K.-W. Park is with the Computer Hacking and Information Security Department, Daejeon University, 62 Daehak-ro, Dong-gu, Daejeon 300-716, South Korea. E-mail: oongbak@dju.kr.*
- *J. Han is with NexR Co. Ltd., #9, Samheung Building, 735-10, Yeoksam-Dong, Gangnam-Gu, Seoul 137-070, South Korea. E-mail: jason.han@nexr.com.*
- *J. Chung is with Intel Labs, Intel Co. Ltd., Santa Clara, CA 95054-1537. E-mail: jaewoong.chung@intel.com.*
- *K.H. Park is with the Electrical Engineering Department, Korea Advanced Institute of Science and Technology, 291 Daehak-ro, Yuseong-gu, Daejeon 305-701, South Korea. E-mail: kpark@ee.kaist.ac.kr.*

Manuscript received 11 July 2011; revised 30 Oct. 2011; accepted 27 Dec. 2011; published online 12 Jan. 2012.
For information on obtaining reprints of this article, please send e-mail to: tsc@computer.org, and reference IEEECS Log Number TSC-2011-07-0067.
Digital Object Identifier no. 10.1109/TSC.2012.1.

mutual agreement. One way of ensuring the integrity and nonrepudiation of a transaction (i.e., where participants cannot deny the context of a billing transaction) is to integrate a public key infrastructure (PKI)-based digital signature [10] into each billing transaction to prevent corruption. Several studies [11], [12], [13] 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 [14].

- *Computation efficiency of a billing transaction*. Cloud service users and CSPs can generate a vast number of billing transactions because on-demand cloud services dynamically scale their capacity upwards or downwards. For example, in the case of *iCubeCloud* [15] (which is the underlying cloud computing platform of this study), the billing frequency per user of the top 15 percent heavy users is typically about 4,200 billing transactions per day. They occasionally generate more than 200 billing transactions per second (e.g., when starting a Hadoop cluster), as they usually invoke massively parallel processes in a bursty manner. The frequent billing transactions lead to excessive computational overhead for both the CSP and the user when the above-mentioned security feature is involved in the billing transaction. Consequently, the overhead imposed by the billing transaction should be within acceptable limits so as to be applicable to a wide spectrum of computing devices, such as smartphones, tablets, netbooks, and desktop PCs.
- *Trusted SLA monitoring*. Once a cloud service user and CSP agree on an SLA, the service quality should be monitored in a trusted manner. A CSP may deploy a monitor and make the monitor's functionality available to its users. However, the presence of the monitor itself is insufficient because the monitors are deployed on cloud resources that are not operated by users [16]. The CSP may deliberately or unintentionally generate incorrect monitoring records, resulting in incorrect bills. To provide an SLA monitoring mechanism, several studies have made great efforts to design solutions that meet various requirements, including scalability with distributed resource monitoring [17], data-flow monitoring [18], and predictions of SLA violations [19], rather than addressing security concerns such as the integrity and trustworthiness of the monitoring mechanism. Thus, they are not fully supportive of the security issues.

A thorough investigation of existing billing systems in various computing environments has helped us identify the above limitations in terms of security capabilities or computational overhead. To overcome these limitations, we propose a secure and nonobstructive billing system termed THEMIS. Specifically, we devised the following three mechanisms, which drive the architecture of our billing system:

- *Support for a mutually verifiable billing mechanism*. We refer the security features of integrity and nonrepudiation to mutual verifiability. Our proposed billing system, THEMIS, introduces the concept of a cloud notary authority (CNA) for the supervision of billing transactions. For each billing transaction, the CNA generates mutually verifiable binding information that can be used to resolve future disputes between a user and a CSP. Consequently, THEMIS can construct credible and verifiable billing transaction, which is likely to be accepted by users and CSPs.
- *A billing mechanism with minuscule computational overhead*. The huge number of billing transactions leads to excessive computational overhead or to a billing system bottleneck. To mitigate these problems, we propose a computationally efficient billing scheme, which replaces the prohibitively expensive PKI operations with a few hash and symmetric key operations while providing mutual verifiability. As a result, the billing transaction overhead is significantly reduced.
- *Support for trusted SLA monitoring*. We devised an SLA monitoring module, called S-Mon, which can be deployed in the computing resources of CSPs. S-Mon has a forgery-resistive monitoring mechanism in which even the administrator of a cloud system cannot modify or falsify the logged data. S-Mon exploits two hardware-based security mechanisms: The trusted platform module (TPM) [20] and the Trusted Execution Technology (TXT) [21]. A more comprehensive description of TPM and TXT is given in Section 4.2. By means of the hardware components above, S-Mon can: 1) monitor the level of SLA achievements with regard to the user's cloud resources; 2) take action when a violation is detected, such as accurately recording a violation in a secure storage region; and 3) deliver the logged data to the CNA after the service session is finished.

The remainder of the paper is organized as follows: In Section 2, we review related works and analyze existing billing systems. In Section 3, we present the overall system design and components of the proposed billing system. In Section 4, we illustrate the proposed billing protocol. In Section 5, we evaluate the performance of the proposed billing system. In Section 6, we discuss the real deployment and additional extensibility of this work. Finally, in Section 7, we present our conclusions.

2 Related Works

Billing systems that track and verify the usage of computing resources have been actively studied and developed in the research area of grid and cloud computing. Many studies have analyzed preexisting billing systems of grid and cloud computing environments [22], [23]. They have tried to identify the new requirements of the shift in the computing paradigm from grid computing to cloud computing. In this section, we briefly discuss experimental results as we evaluate existing billing systems in terms of their security level and billing overhead. We evaluate the billing systems in an identical computing and network environment. A more comprehensive evaluation of the experimental results and the experimental environment can be found in Section 5.

2.1 Billing Systems with Limited Security Concerns

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 [24], [25]. Several researchers presented a resource usage processing system [6], [7], [8], [9] for recording the usage of grid resources. Fig. 1a shows the architecture and characteristics of a billing system with limited 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 billing agent. APEL [6] presents a billing system that processes log information to create quantified accounting records. Other resource management and billing frameworks that were suggested as part of traditional grid approaches: Namely, Condor/G [7], GRASP [8], and Tivoli [9].

However, rather than addressing security concerns, they focus on notions such as distributed resource usage metering and an account balancing mechanism for a distributed grid environment. Thus, they cannot provide transaction integrity, nonrepudiation, and trusted SLA monitoring, even though they had a nonobstructive billing transaction latency of 4.06 ms in our experimental environment. These security functions are precluded because the frameworks were designed for a distributed grid environment, not for a pay-per-use billing scheme.

2.2 Security-Enhanced Billing Systems

Several electronic payment schemes have been proposed in the literature in an attempt to provide security-enhanced billing mechanisms. They include micropayment-based schemes such as MiniPay [26] and NetPay [27]. 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 billing transactions by releasing a certain number of hashes in the hash chain. On the basis of the micropayment-based scheme, Pay-as-you-Browse [28] and XPay [29] incorporated the micropayment concept into distributed computing frameworks and cloud-hosted services. As shown in Fig. 1b, the micropayment-based scheme has a short billing latency (4.70 ms) in our experimental environment. However, it cannot support the security features of nonrepudiation and trusted SLA monitoring because micropayment schemes are mainly designed for transaction integrity rather than other security features.

Research on cloud or grid computing has developed the following market models and PKI-enhanced billing and accounting frameworks: DGAS [11], SGAS [12], and GridBank [13]. They have a secure grid-wide accounting and payment handling system in which each user's accounts and resource usage records are maintained with a PKI-based digital signature. The commercial cloud services of Amazon EC2, S3 [1], and Microsoft Azure [2] provide users with a service usage report via secure communication and monitoring tools such as CloudWatch [30]. Yet, the CSPs have not been adopting transparent utility-type pricing models for their SLAs.

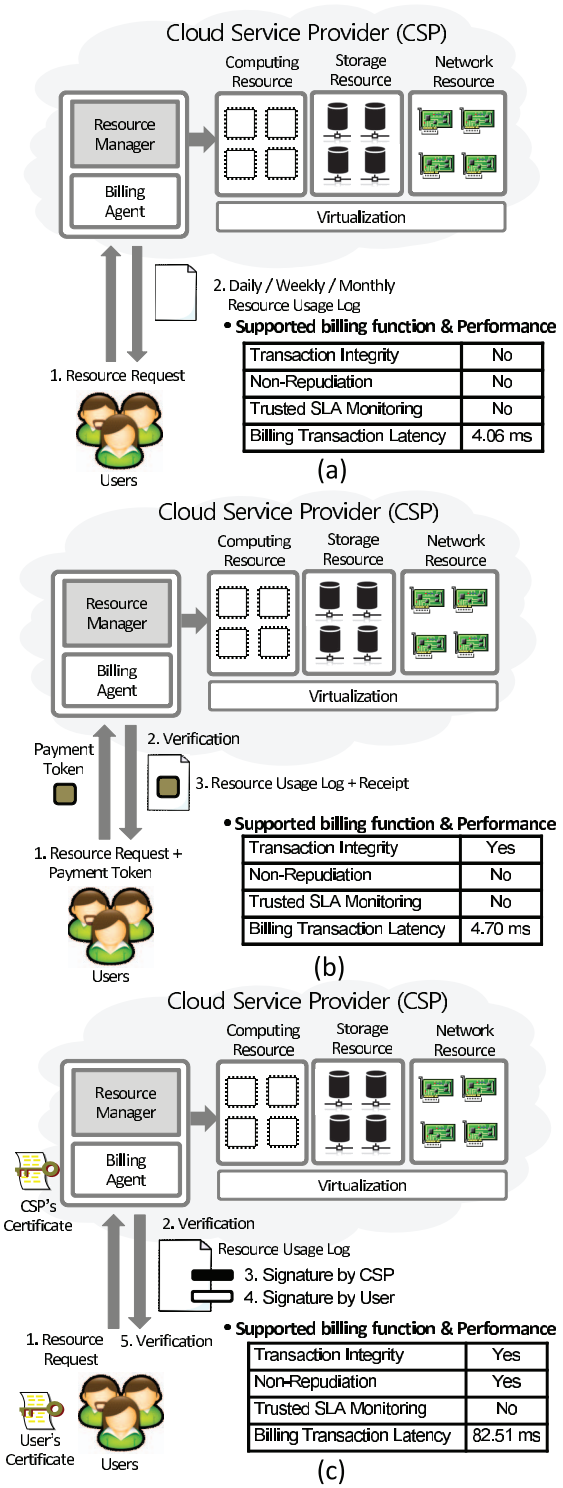


Fig. 1. Brief overview of billing systems and its characteristics in terms of billing function and transaction latency: (a) billing system with limited security concerns, (b) billing system with micropayment, and (c) PKI-based billing system.

Fig. 1c illustrates the organization of a PKI-based billing system and its characteristics in terms of the security level and billing overhead. It has a longer billing latency (82.51 ms) than the other systems in our experimental environment. The extent of the overhead is mainly determined by the extremely high complexity of the RSA [31] operations when the PKI is used for a billing system

System	Transaction Integrity	Non-Repudiation	Trusted SLA Monitoring	Billing Latency
Billing System with limited Security	No	No	No	Avg. 4.06ms
Micropayment-based Billing System	Yes	No	No	Avg. 4.70 ms
PKI-based Billing System	Yes	Yes	No	Avg. 82.51ms
THEMIS	**Yes**	**Yes**	**Yes**	**Avg. 4.89 ms**

- **Experiment Environment:**
  Server-Side: (CPU: X5650, RAM: 24GB), User-Side: (CPU: Z530, RAM: 1GB)

Fig. 2. Summary of relevant works in terms of security functionality and billing latency.

by a thin client or a heavily loaded server. The computational overhead can be a severe drawback when a number of cloud service users and the CSP generate a vast amount of billing transactions.

2.3 Summary

Fig. 2 shows the security functions and billing transaction latency values of the above-mentioned works. The billing systems with limited security concerns and the micropayment-based billing system require a relatively low level of computational complexity: The nonobstructive billing transaction latency is 4.06 ms for the former and 4.70 ms for the latter. Nevertheless, these systems are inadequate in terms of transaction integrity, nonrepudiation, and trusted SLA monitoring.

Despite the consensus that PKI-based billing systems offer a high level of security through two security functions (excluding trustworthy SLA monitoring), the security comes at the price of extremely complex 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.

The last row clarifies the design goal of THEMIS and the final value of the billing transaction latency. THEMIS comply with the system requirements mentioned above and are based on three main principles, namely, mutually verifiable billing with a CNA, nonobstructive billing transactions (4.89 ms), and trusted SLA monitoring.

From the viewpoint of SLA monitoring, several studies [17], [18], [19] have made great efforts to design solutions that meet various requirements. These works focus on the notions of scalability with a distributed resource monitoring [17], a data flow monitoring [18], and a prediction of SLA violations [19], rather than addressing security concerns such as the integrity and trustworthiness of a monitoring mechanism. When the above monitoring techniques are used in conjunction with THEMIS, the extensibility of this work can be improved further.

This study is an extension of our previous work [32], in which we focused on the protocol design of a mutually verifiable billing system. Our objective in this study, however, is to devise a trusted SLA monitoring module and integrate the overall components in *iCubeCloud*, a real cloud computing platform [15]. THEMIS act as the key primitive for secure, trustworthy billing transactions.

3 Design of THEMIS Billing System

We present an overview of the THEMIS billing system in this section. We first introduce the important components of THEMIS and then describe the overall billing process.

3.1 The Proposed THEMIS Infrastructure

Fig. 3 shows the overall architecture of THEMIS billing system. The four major components of the architecture are listed as follows:

- *CSP*. 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.
- *Users*. 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.
- *CNA*. 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

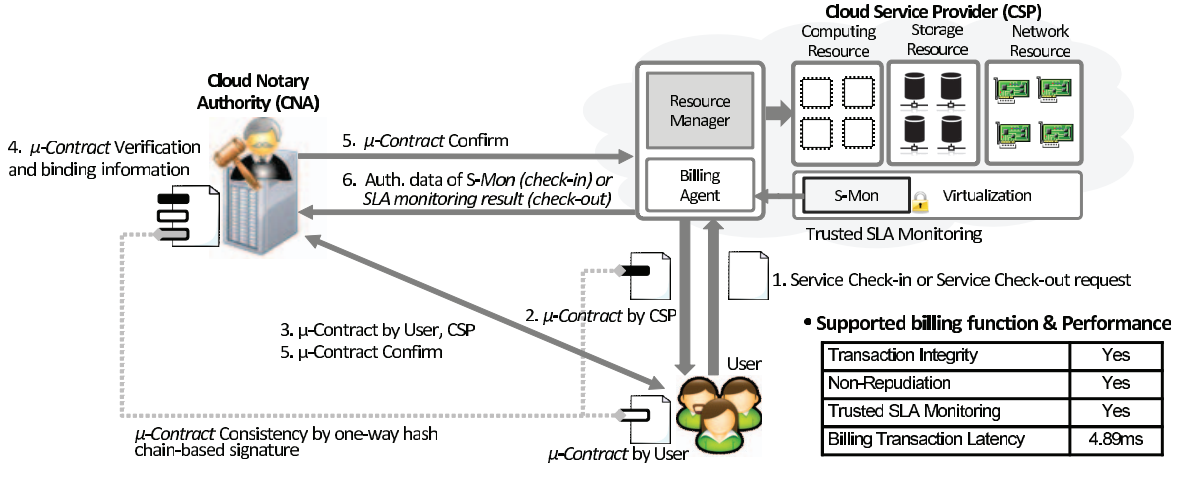


Fig. 3. Overall architecture and flowchart of the billing transactions of THEMIS.

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.

- *Trusted SLA Monitor (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. 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.

3.2 Overall Billing Process of THEMIS

After a registration phase, THEMIS can use the above components to provide a mutually verifiable billing transaction without asymmetric key operations of any entities. The registration phase involves mutual authentication of the entities and the generation of a hash chain by each entity. The hash chain element of each entity is integrated into each billing transaction on a chain-by-chain basis; it enables the CNA to verify the correctness of the billing transaction. In addition, S-Mon has a forgery-resistive SLA measuring and logging mechanism. THEMIS consequently supervises the billing, and because of its objectivity, it is likely to be accepted by users and CSPs alike.

The billing transactions can be performed in two types of transactions: A service check-in for starting a cloud service session and a service check-out for finalizing the service session. These two transactions can be made in a similar way. Each billing transaction is performed by the transmission of a message, called a μ-*contract*. A μ-*contract* is a data structure that contains a hashed value of a billing context and the hash chain element of each entity. With the sole authority to decrypt both the μ-*contract* from the CSP and the μ-*contract* of the user, the CNA can act as a third party to verify the consistency of the billing context between the user and the CSP.

Fig. 3 shows the overall process of the billing transaction with our billing system. The main steps are as follows:

1. The user generates a service check-in or check-out request message and sends it to the CSP.
2. The CSP uses an element from the CSP's hash chain to send the user a μ-*contract-CSP* as a digital signature.
3. The user uses an element from the user's hash chain to generate a μ-*contract-User* as a digital signature. The user then combines the μ-*contract-User* with μ-*contract-CSP* and sends the combined μ-*contract* to the CNA.
4. The CNA verifies the μ-*contract* from the user, and generates mutually verifiable binding information of the user and the CSP to ensure the consistency of the μ-*contract*.
5. The billing process is completed when the user and the CSP receive confirmation from the CNA.
6. Finally, in the case of a service check-in, the S-Mon of the user's cloud resource transmits authentication data of the S-Mon to the CNA. In the case of a service check-out, S-Mon sends a report of the SLA monitoring results to the CNA.

A more comprehensive description of the above transaction can be found in Section 4.

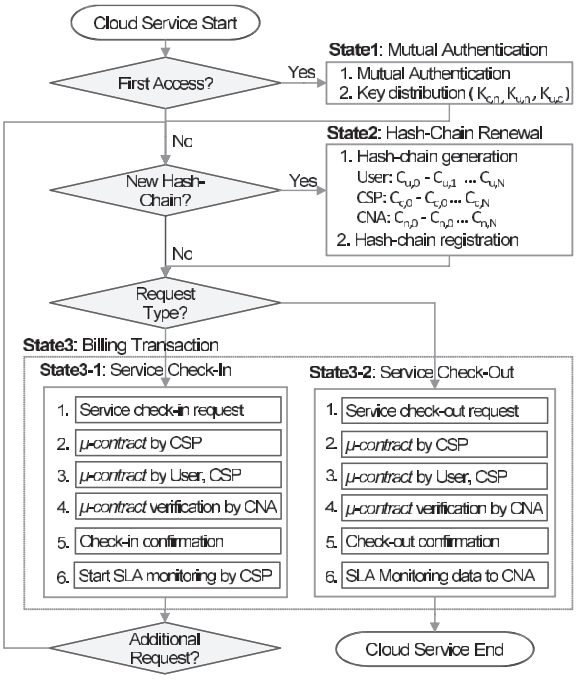


Fig. 4. Flow diagram of the proposed billing protocol.

4 Proposed Billing Protocol

In this section, we describe the overall transactions of the proposed billing scheme, and in Section 4.4, we analyze the security and safety of the proposed billing system.

4.1 Description of the THEMIS Billing Protocol

Fig. 4 shows a flow diagram of the overall transactions of the proposed billing protocol. The protocol consists of the three states.

State 1 involves key sharing and *State 2* involves the generation of a hash chain. Together these states serve as the initial step of future billing transactions. They help optimize the computational overhead of the billing mechanism and facilitate mutual verifiability.

State 3 is for an actual billing transaction. During the billing transaction, the user and the CSP both generate a μ-*contract*. The μ-*contract* is notarized by the CNA. The use of the μ-*contract* means that the billing transaction replaces the prohibitively expensive PKI operations to a few hash and symmetric key operations without compromising the security level of the PKI; as a result, the billing transaction overhead is significantly reduced. In addition, S-Mon exchanges data for trusted SLA monitoring. Table 1 lists the notations of the entities and messages of the proposed protocol. The details of the three states are as follows:

State 1 (mutual authentication). This state is for a user who accesses the CSP for the first time. When the user first accesses the CSP, 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: $K_{u,c}$.

TABLE 1
Notation of the Entities and Messages

Definition of the entity symbols
c: Cloud Service Provider (CSP)
u: User
n: Cloud Notary Authority (CNA)
m: SLA Monitoring Module (S-Mon
Definition of the Message symbols
$K_{\alpha,\beta}$: Shared Key between α and β
PK_α : Public Key of α
SK_α : Private Key of α
$H(M)$: Hash result for message M
Hash Chain: C_0 - C_1 - C_2 - . . . - Cn , $H(C_n) = C_{n-1}$ (Usage sequence →; Chain generation sequence ←)
$C_{\alpha,n}$: nth element of the hash chain of α
T_s : Time-Stamp
N_α: Nonce value for preventing replay attack by α
S : Stipulation context of billing transaction
$\{M\}K$: M encrypted by K
$\{M\}PK$: M encrypted by public Key
$\{M\}SK$: Digital signature for M by private key

State 2 (hash chain generation). This state is for generating and registering a hash chain among the CSP, the CNA, and the user. Each of these three parties generates 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. As shown in Fig. 5, 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 signed hash chain 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, the elements of the hash chains are used chain by chain to generate a μ-*contract* for future billing transactions. *State 2* is skipped until the corresponding hash chain is depleted.

State 3 (billing transaction). An actual billing transaction is performed at the beginning of *State 3*. In this state, a user can perform two types of billing transactions: "a service check-in" (to start a service session, State 3-1) and "a service check-out" (to end a service session, State 3-2). The service check-in is for requesting a new cloud service, such as a virtual machine service or a storage service. A user who wants to end the cloud service can perform a billing transaction for "a service check-out."

Both types of transactions can be performed in a similar way. The difference between them is the context of the message. The context of a service check-in includes the SLA and data for the initialization of SLA monitoring. The context of the service check-out includes information that can be used to verify data from the SLA monitoring module, S-Mon.

- *State 3-1 (billing transaction for a service check-in)*. As shown in Fig. 5, a user who intends to receive a cloud service from a CSP sends a service check-in request message (*Message 3-1*) to the CSP. Upon receiving the message, the CSP transmits a stipulation (S) and a μ-*contract-CSP* to the user. The S contains a service invoice and an SLA that covers guaranteed performance factors, such as availability, CPU speed, I/O throughput, a time stamp, and the price. The μ-*contract-CSP* contains a hash chain element of the CSP, $C_{c,n}$. The hash chain element, which is listed in Table 1, is updated for each μ-*contract-CSP* on a chain-by-chain basis so that all of the μ-*contract-CSP* can be linked and verified sequentially toward the seed value ($C_{c,0}$) of the hash chain.

  After receiving the μ-*contract-CSP* (*Message 3-2*), the user generates a notary request message (*Message 3-3*, which is a combination of the μ-*contract-CSP* and the μ-*contract-User*) and sends it to the CNA. When *Message 3-3* 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 billing request message. We note that only the CNA can acquire both the $H(S)$ section of the μ-*contract-CSP* and the $H(S)$ section of the μ-*contract-CSP*. If the two contexts are identical to each other, the CNA sends a confirmation message (*Message 3-4*) to the user and

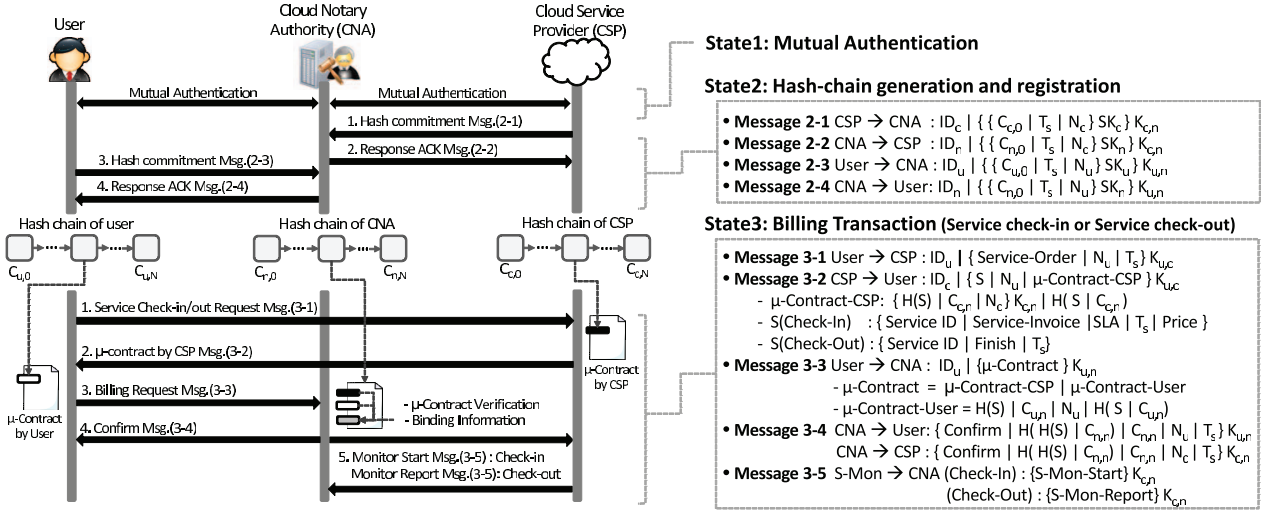


Fig. 5. Overall message transaction of THEMIS from *State 1* to *State 3*.

the CSP. Otherwise, the CNA sends an error message to the user and the CSP. The billing transaction is completed when the user and the CSP receive the confirmation message.

In this transaction, the *μ-contract-CSP* contains the output of the hash function with two input values (S and $C_{c,n}$). Due to the NP-completeness of the hash function, only the CSP can perceive the $C_{c,n}$. Similarly, only the user and the CNA can perceive the $C_{u,n}$ value of *μ-contract-User* and the $C_{n,n}$ value of the confirmation message, respectively. This method enables the CNA to generate information that is binding among all the involved entities. The verified *μ-contract* is subsequently retained on the local repository of the CNA for future accusations. This is the end of the billing transaction of the service check-in.

Following that, the S-Mon of the user's cloud resource sends a monitor start message (*Message 3-5*) to the CNA so that the service session is started. Section 4.2 describes in more detail how the SLA is monitored and verified.

- *State 3-2 (billing transaction for a service check-out)*. A user who intends to end the cloud service of the CSP performs a billing transaction that is similar to the service check-in transaction. The main difference between the check-in and check-out transactions is that S-Mon sends an SLA monitoring result to the CNA in the confirmation message (*Message 3-5*). The CNA can consequently determine whether the SLA has been violated. If the CSP is unable to meet the SLA, the CNA may impose penalties, such as reducing or canceling the payment.

4.2 S-Mon: SLA-Monitor

To provide trusted SLA monitoring, we devised S-Mon, which can be deployed into computing resources of the CSP. S-Mon provides a forgery-resistive SLA measuring and logging mechanism in a black-box (BB) manner. Thus, even the administrator of the CSP cannot modify or falsify the logged data.

S-Mon is tightly coupled with the billing protocol described in previous section. First, S-Mon is initialized and verified during the service check-in transaction (via *State 3-1*). Second, during the service session, S-Mon monitors the level of SLA achievement with regard to the user's cloud resources. Third, S-Mon reports the SLA monitoring result to the CNA upon the service check-out transaction (via *State 3-2*). Finally, by measuring the service interval and verifying the monitoring result, the service session is confirmed with the above transactions. Thus, the billing transactions become more objective and acceptable to users and CSPs due to the provision of the trusted and forgery-resistant SLA monitoring mechanism of S-Mon.

S-Mon has two hardware-based mechanisms: The TPM [20] and the TXT [21]. The TPM is a widely deployed security chip in commercial-off-the-shelf computer systems. It is designed for the purpose of secure storage and remotely determining the trustworthiness of a platform. TXT is a set of technology by Intel. It provides a secure execution mechanism called a measured launch environment (MLE); it enables a verified execution code in a secure memory region. S-Mon uses the following fundamental technologies:

- *Platform integrity measurement*. To ensure the trusted execution of S-Mon, we utilize a TPM. One of the important features of the TPM is a set of platform configuration registers (PCRs). The PCRs are a set of built-in registers that can be used to store the 160-bit hash values obtained from the SHA-1 hashing algorithm. The PCR values can only be changed by the *Extend()* function, which is an internal function of the TPM. It outputs a hash result with (input value + current PCR value), and then replaces the current PCR value with the output of this operation.

  The MLE uses the PCR's characteristics. Before handling over the control to a program to be executed, the MLE uses the *Extend()* function to extend the resultant value into a PCR. MLE can compare the PCR value with a reference value to ensure that only a verified execution code is invoked in a secure memory region.

  To enable the CNA to verify the platform status, the TPM provides a *Quote()* function, which uses a TPM private key called an attestation identity (*AIK*) to return a digital signature of the current PCR values. The *AIK* is created inside the TPM and protected by the TPM so that *Quote()* provides proof that the output of *Quote()* was generated on the platform.
- *Secure storage with the TPM*. The TPM provides a means of storing data in a secure fashion. The *Seal()* function encrypts the input data with a TPM key and specified PCR values. The *Unseal()* function decrypts the encrypted data only when the specified PCR values and the key are matched with the status of sealing [33]. S-Mon uses the *Seal()* and *Unseal()* functions to protect the SLA monitoring data in such a way that the data can only be decrypted by S-Mon itself.
- *Execution integrity with the TPM*. The TPM has built-in support for a monotonic counter. The increments of this type of counter are in single steps, and the value of the counter is only incremented by the *IncrementCounter()* function. In addition, the TPM has a mechanism that creates a signature of the current tick value of the TPM. The tick data include a signature of the current tick value and its update cycle. These functions are utilized in our verification mechanism. The verification mechanism enables the CNA to determine whether the S-Mon has been executed without a block or a data loss; it also determines when SLA violations occur with the tick value.

We incorporated these fundamental technologies into a three-phase procedure of forgery-resistive SLA measuring and logging. Fig. 6 shows the overall procedure of S-Mon. Each cloud resource has a trusted boot module, called Tboot [34] as an underlying security module. Tboot is an open source module that supports MLE functionality; it enables S-Mon to be executed in a trusted environment. Consequently, only the CNA's registered software (hypervisor, dom0, and S-Mon) can be invoked [35].

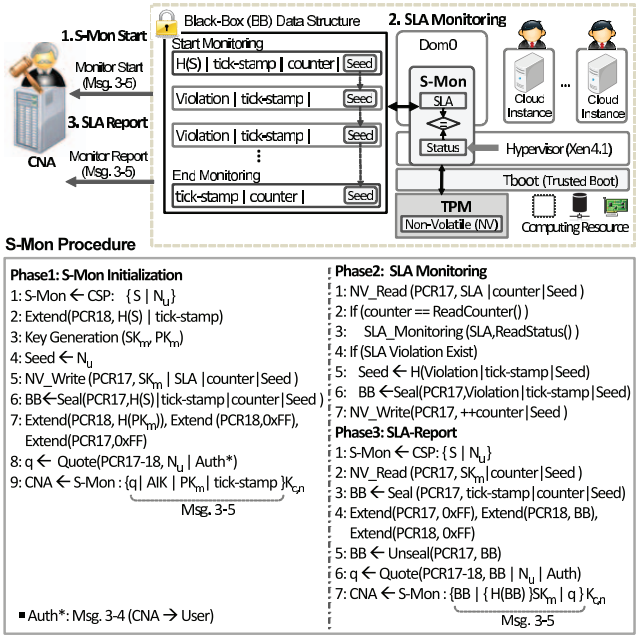


Fig. 6. Overall process and message transactions of S-Mon interacting with CNA.

The S-Mon procedure consists of three phases. Phase1 is performed for the beginning of a service session; Phase2 is periodically invoked during a service session; and Phase3 is performed for the end of the service session. The CNA can consequently determine whether the SLA has been violated. We note that S-Mon deliver the logged data to the CNA only after the service session is finished or when the user requires SLA monitoring data from the CNA via Phase3. The details of the three S-Mon phases are as follows:

- *Phase1 (S-Mon initialization)*. In a billing transaction for a service check-in, the S-Mon of the user's cloud resources initializes itself by accepting S, which contains the SLA to be monitored and N_u (line 1). To enable the CNA to check the freshness of the S-Mon, S-Mon performs an *extend()* operation by inputting $H(S)$ and a tick-stamp (line 2). The tick-stamp confirms the starting time of S-Mon.

  For secure communication between S-Mon and the CNA, S-Mon generates a pair of private and public keys, SK_m and PK_m (line 3). The *counter* of line 5 is the value of the monotonic counter of S-Mon's TPM. The *counter* and $Seed$ (initial value $= N_u$) are used in Phase2 and Phase3 for the integrity check of the data logged by S-Mon.

  The initial data (SK_m, SLA, counter, $Seed$, and the tick-stamp) are sealed in a nonvolatile region (NV) of the TPM and *BB* data structure by the *seal()* operation (lines 5-6). The NV represents areas of flash storage inside the TPM itself, and the BB is a repository for storing all the data logged by S-Mon. The NV and the *BB* can be accessed only when the corresponding PCR value is matched with the status of the sealing. This limitation enables S-Mon to store persistent data in such a way that only S-Mon itself can access the data.

  S-Mon subsequently stores the current status of S-Mon in q by using the *Extend()* and *Quote()* operations (lines 7-8). The $Extend(PCR, 0xFF)$ operation prevents any other software from accessing the NV and *BB*. Finally, S-Mon transmits a message (*Message 3-5*) to start the SLA monitoring (line 9).

  Message 3-5 contains data on the freshness of S-Mon as well as the authentication data (*Auth*) of the user's cloud resources. The message enables the CNA not only to ensure that the S-Mon was initialized with the correct parameters (S, N_u, and the tick-stamp) but also to verify that the target resources to be monitored by the S-Mon are correctly bound to the cloud resources of the user. The *Auth* comprises the context of *Message 3-4* between the CNA and the user, which is encrypted with $K_{u,n}$ so that the only the user and the CNA can decrypt the message. During the service check-in phase (via *State 3-1*), the user enters the *Auth* into the user's cloud resource as a log-in parameter, and the data are delivered to the S-Mon by means of Xen-hypercall [36]. The S-Mon then includes the data in *Message 3-5*. Finally, the CNA can verify the freshness and correctness of the S-Mon by comparing the context of *Message 3-5* with the expected value.
- *Phase2 (SLA monitoring)*. Phase2 periodically occurs during the service time. To ensure its own execution integrity, S-Mon performs this phase only when the counter value stored in the NV is the same as the monotonic counter value of the TPM (lines 1-3). Whenever this phase occurs, the monotonic counter is increased, and the value of the counter is stored in the NV (line 7). Thus, when Phase2 is executed again, the counter value in the NV must be the same as the value of the monotonic counter. Otherwise S-Mon is aborted, which means the counter value was tampered with or the unsealed data were stale.

  To detect the current system status, we implemented a back-end driver deployable into a hypervisor to read the system status of the user's cloud resources. The driver uses Xen-hypercall [36] to read the system status details of the CPU time, memory, and network status (line 3) and it records the violation in the *BB* whenever a violation is detected (line 4-6).

  Whenever a violation is detected, the violation context, the extended *Seed* value (line 5), and the current tick-stamp are appended to the *BB*. Each violation context and tick-stamp is bound by the *Seed*. Because the *Seed* value is extended by a hash function for every SLA violation, each violation context is linked to the previous violation context. This linking process enables the CNA to check the consistency of the *BB* context.
- *Phase3 (SLA Report)*. Phase3 is executed when the corresponding service session is ended by the user. S-Mon transmits the *BB*, which contains the SLA monitoring result to the CNA. Before sending the *BB*, S-Mon appends the final tick-stamp, the counter, and *Seed* (line 3). S-Mon stores the current status itself by using the *Extend()* operation (line 4). A digital signature of TPM is used to bind *BB*, N_u, and *Auth*

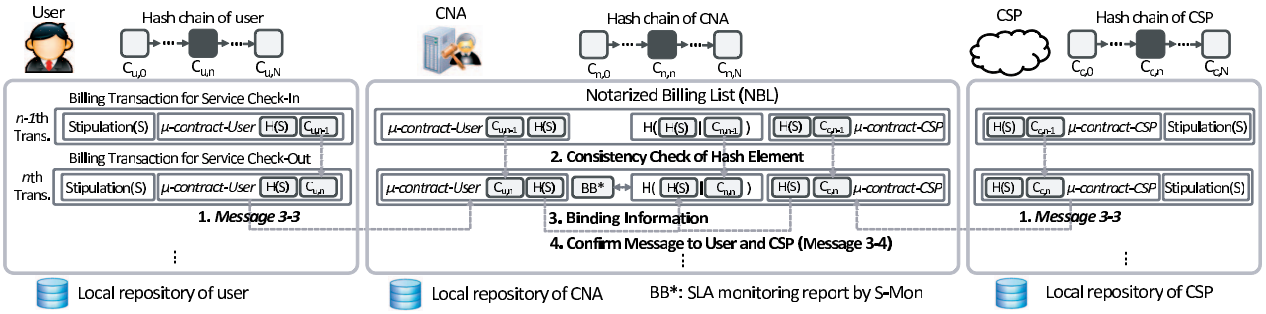


Fig. 7. Overall message transactions and local repositories for mutually verifiable billing transactions.

to q with the PCR values (line 6). Finally, S-Mon returns the results with its digital signature by SK_m to the CNA (line 7). The context of *BB* enables the CNA to check whether S-Mon was executed correctly without a break or halt and whether the returned result was truly generated by S-Mon.

4.3 Verification of the Billing Records

This section elaborates how the billing can be verified in collaboration with the CNA. As shown in Fig. 7, when *Message 3-3* arrives at the CNA during a service check-in or check-out transaction, the CNA checks the consistency of the hash chain elements ($C_{u,n}$ and $C_{c,n}$) from the user and the CSP by checking the link with the previously used hash chain elements ($C_{u,n-1}$ and $C_{c,n-1}$). Next, the CNA checks the consistency between the $H(S)$ of *μ-contract-User* and the $H(S)$ of *μ-contract-CSP*.

The checking process enables the CNA to verify whether the CSP and user have the same stipulation (S). The CNA subsequently generates binding information, which contains the hashed value of $H(S)$ and the hash chain element ($C_{n,n}$) of the CNA. The CNA then sends the user and the CSP a confirmation message with the binding information (*Message 3-4*). Upon receiving the message, the user and the CSP confirm that the corresponding billing transaction (the *μ-contract*) is correctly notarized. In the case of the billing transaction for a service check-out, the S-Mon of the user's cloud resources transmits the *BB* via *Message 3-5*. After each billing transaction, the CNA retains the corresponding binding information and the *BB* at the CNA's local repository in a type of notarized billing list (NBL). The NBL is an XML-based data structure for storing evidence of the billing transactions for future accusations. All of the contexts are periodically stored with the digital signature of the CNA to ensure the integrity of the NBL context. The user and the CSP also store the billing-related information as evidentiary data.

Fig. 8 illustrates how the NBL is used to prove the integrity of certain billing transactions by the verification module of the CNA. The verification module has 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 the CSP by comparing the stipulation with the binding information of the CNA. In addition, the CNA-Verifier can check the correctness of the *BB* by comparing the $H(S)$ of the NBL with the $H(S)$ of the *BB*. The User-Verifier and the CSP-Verifier check the correctness of a billing transaction asserted by the user and the CSP, respectively.

For example, if a CSP asserts that a user repudiates a certain billing 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 uses the CNA-verifier to verify the claim. If the claim is correct, the CNA then demands to see the stipulation (S) used to generate the *μ-contract-User*. The CNA uses the User-Verifier and the CSP-Verifier to derive the hash value. Any discrepancy between the output of the hash function and the stored data of the NBL proves that either the user or the CSP has modified the stipulation of the relevant billing.

4.4 Case Study on Potential Attacks against THEMIS

In this section, we analyze the security and safety of the proposed billing system. Our analysis is based on consideration of falsified data, replay attacks, and man-in-the-middle (MITM) attacks. On the one hand, we assume that the underlying cryptography and hardware-based security mechanisms (TPM and TXT) are invulnerable in terms of data confidentiality and integrity; hence, we ignore attacks with cryptanalysis and physical attacks against the system's CPU or TPM. One the other hand, any principle can place or inject data on any storage device and link at any time. In addition, any principle can see, delete, alter, and redirect all exchanged messages or replay messages recorded from past communications.

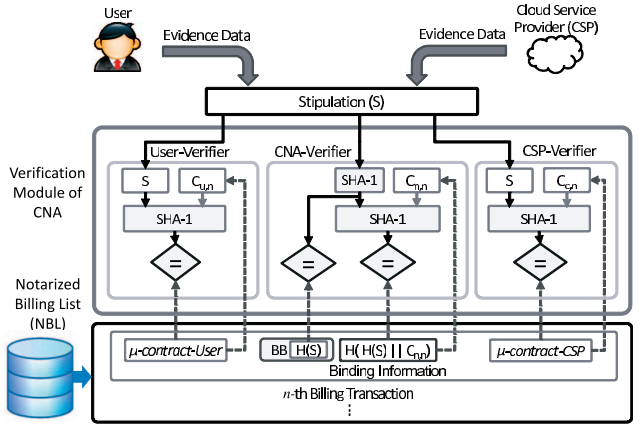


Fig. 8. Verification mechanism with the NBL in the CNA.

Case 1. Falsified μ-contract *attacks by a malicious CSP or malicious user:*

1. Forgery of the *μ-contract* by a malicious CSP or a malicious user before the confirmation of a billing transaction.
2. Forgery of the *μ-contract* by a malicious CSP or a malicious user after the confirmation of a billing transaction.

Defense mechanism:

- In the case of 1, a malicious CSP or user can try to falsify the *μ-contract* before the corresponding billing transaction is confirmed. However, the CNA compares the hash chain element of the *μ-contract-CSP* with that of the *μ-contract-User*. If these elements do not match, the CNA does not send a confirm message to the CSP and the user.
- In the case of 2, a malicious CSP or user can try to falsify the *μ-contract* after the confirmation of the billing transaction. However, the CNA records the *μ-contract* in the NBL. Hence, the CNA can identify the malicious CSP or user by comparing the $H(S)$ and C_n of the NBL with the falsified *μ-contract*.

Case 2. Falsified BB attacks or system modification by a malicious CSP:

1. Forgery of the *BB* or system modification by a malicious CSP during a service session to induce inaccurate SLA monitoring.
2. Forgery of the *BB* by a malicious CSP after the service session.

Defense mechanism:

- In the case of 1, a malicious CSP can try to falsify the context of the *BB* during the corresponding service session. However, the *BB* can be accessed only by S-Mon because the TPM seals the *BB* with the PCR values of the trusted S-Mon. In addition, a malicious CSP can try to modify the system layer to induce inaccurate SLA monitoring. However, Tboot invokes only registered software, namely, hypervisor, dom0, and S-Mon. Any invocation of the modified hypervisor or dom0 or S-Mon is blocked because the PCR values of the TPM fail to match the known PCR values of the predefined trusted software set.
- In the case of 2, a malicious CSP can try to falsify the context of the *BB* after the service session. However, the context of the *BB* contains its digital signature by a private key (SK_m), which can be accessed only by S-Mon. Furthermore, all the items of logged data in the *BB* are linked to each other in a hash chain manner so that any modification or deletion of the *BB* can be detected.

Case 3. Replay and MITM attacks:

1. A replay attack for THEMIS-based billing transaction messages.
2. An MITM attack for THEMIS-based billing transaction messages.

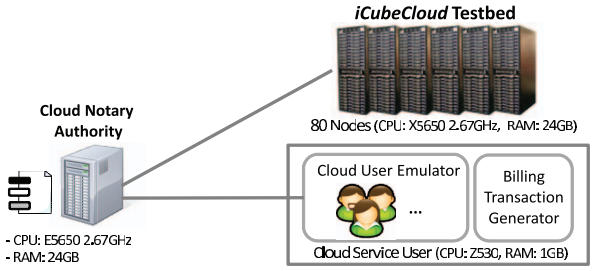


Fig. 9. Experiment environment for measuring the performance of THEMIS.

Defense mechanism:

- In the case of 1, An intruder can try to attack by sending captured messages recorded from past billing transactions. However, all the messages contain nonce data and a time stamp in this billing protocol to verify the freshness of the data.
- In the case of 2, the CSP and user generate billing transaction messages that are altered for each billing transaction, and the messages are encrypted and transmitted by means of the shared keys that are paired in a previous state. Thus, the intruder cannot successfully masquerade as the user or the service device.

5 Performance Evaluation

In this section, we present the performance results of our prototype version of THEMIS. First, we demonstrate the overall experimental environment. We then describe the operational efficiency of the billing protocol to evaluate the performance of THEMIS in terms of latency and throughput. Finally, we present the performance overhead of S-Mon with respect to the cloud computing platform.

5.1 Experimental Environment

Fig. 9 shows the overall experimental environment. To evaluate the performance characteristics of THEMIS, we constructed a cloud user emulator and coupled it to a billing transaction generator.

The user emulator is connected to THEMIS, and the emulator receives control signals from the billing transaction generator. The generator is a module that generates control signals to produce billing request messages. The user emulator has an Intel Z510 processor and a 1-GB main memory; the other components, including the CNA, the billing agent of the CSP, and the cloud computing platform, have a Xeon X5650 processor and a 24-GB main memory.

5.2 Billing Protocol Efficiency and Comparative Evaluation

The performance of the billing protocol in terms of the consumption of processing and communication resources is an important factor to be considered when designing billing protocols. First, we analyze how THEMIS compares with other billing schemes in terms of computation and communication efficiency. To compare the overall latency of each billing transaction scheme, we measured the billing transaction latency, which is based on the interval between the start

Billing System	Security Functionality			User				CSP				CNA or 3rd party				Communication	Total Transaction Time
	Mutual Verifiability	Integrity	Trusted SLA Monitoring	RSA PU	RSA PR	AES	SHA	RSA PU	RSA PR	AES	SHA	RSA PU	RSA PR	AES	SHA		
Limited Security	No	No	No	0	0	2	0	0	0	2	0	0	0	0	0	Avg.3.752ms	Avg.4.06ms
				Avg. 0.270 ms				Avg. 0.042 ms				Avg. 0.0 ms					
Micropayment	No	Yes	No	0	0	4	0	0	0	4	1	0	0	0	0	Avg. 4.231ms	Avg. 4.70ms
				Avg. 0.404 ms				Avg. 0.065 ms				Avg. 0.0 ms					
PKI-based Billing	Yes	Yes	No	2	1	3	0	1	2	3	0	1	0	0	0	Avg.13.81ms	Avg.82.51ms
				Avg. 59.807 ms				Avg. 6.881 ms				Avg. 2.01 ms					
THEMIS (Initial Reg.)	Yes	Yes	Yes	1	1	1	n	1	1	1	n	2	1	1	n	Avg. 14.28ms	Avg. 93.05ms
				Avg. 65.893 ms				Avg. 6.431 ms				Avg. 6.451 ms					
THEMIS (Billing)				0	0	4	2	0	0	3	1	0	0	4	2	Avg. 4.379ms	**Avg. 4.89ms**
				Avg. 0.402 ms				Avg. 0.049 ms				Avg. 0.065 ms					

- Limited Security: *iCubeCloud* with traditional approach
- THEMIS: *iCubeCloud* empowered with proposed billing scheme

* PU: Public key operation, PR: Private key operation

Fig. 10. The number of public/private keys and the symmetric key operations with the total operation time for each billing schemes.

and end of each billing scheme. In the case of THEMIS, we measured the interval between the starting time of *Message 3-1* and the ending time of *Message 3-4* on the client side. Because *Message 3-5* is related to the S-Mon after the confirmation of the billing transaction, the interaction can be interleaved with the other message transaction.

Fig. 10 shows the number of public and private keys (RSA 2,048 bits), the symmetric key (AES 256 bits), and the hash (SHA-1) operations performed with the total operating time for each billing scheme. The operating time of each entity is estimated for each billing transaction so that we can measure how much the cryptography contributes to the billing overhead. We note that the keywords "Limited Security" and "THEMIS" in Fig. 10 are the experimental results of *iCubeCloud* with a traditional approach and *iCubeCloud* empowered with THEMIS, respectively. In the case of the billing system with limited security (as described in Section 2.1) and the micropayment (as described in Section 2.2), the billing transaction can be accomplished without asymmetric key operations. These results confirm that even though they have a shorter billing latency (4.06, 4.70 ms) than the others, it fails to meet our security requirement. Despite having a smaller number of cryptography operations per billing, the PKI-based billing scheme (as described in Section 2.2) has a long billing latency (82.51 ms) because it has a certain number of private and public key operations for all of the entities.

In the case of THEMIS, a user who accesses the CSP for the first time or needs the hash chain renewal is asked to perform *State 1* or *State 2*, which requires an asymmetric key operation and multiple hash operations. The authentication and hash chain generation time of THEMIS (93.05 ms) is slightly higher than the operating time of PKI-based billing. However, after the operations, the user can perform a billing operation by processing only four symmetric key operations and two hash operations without compromising the PKI security level. THEMIS's total billing transaction time (4.89 ms) is much shorter than that of PKI-based billing.

From a communication perspective, PKI-based billing transactions and the initial registration process of THEMIS have a longer communication delay than the others because they need to transmit each entity's certificate. In a network environment with a low bandwidth and long delays, the billing latency is affected by the communication overhead. Because THEMIS has a much smaller communication overhead (4.379 ms) than the PKI-based billing system (13.81 ms), THEMIS can provide a shorter billing latency than a PKI-based billing system in the limited network environment.

5.3 Throughput Evaluation

Fig. 11a illustrates how the throughput of the billing transactions mutates as the number of billing requests per second varies. The number of billing requests per second ranges from 1,000 to 15,000. For the PKI-based billing protocol, we found that the throughput was saturated on 903.3 transactions per second as the number of billing requests increased. This outcome is due mainly to the cryptography operations and the communication overhead of both the client side and the server side. In the case of the billing system with limited security concerns as described in Section 2.1, the throughput is saturated on 12,088.1 transactions per second. This outcome is due to their lower computation and communication overhead than the other systems. In the case of THEMIS and the micropayment, the throughput is saturated on 10,680.9 and 10,770.4 transactions per second, respectively, as the number of billing requests increases. This phenomenon is due to the fact that the quantity of THEMIS and micropayment operations of the user and server provider is much smaller than that of PKI-based billing. This result confirms that the THEMIS billing protocol can seamlessly provide a nonobstructive billing transaction whenever the number of requests per

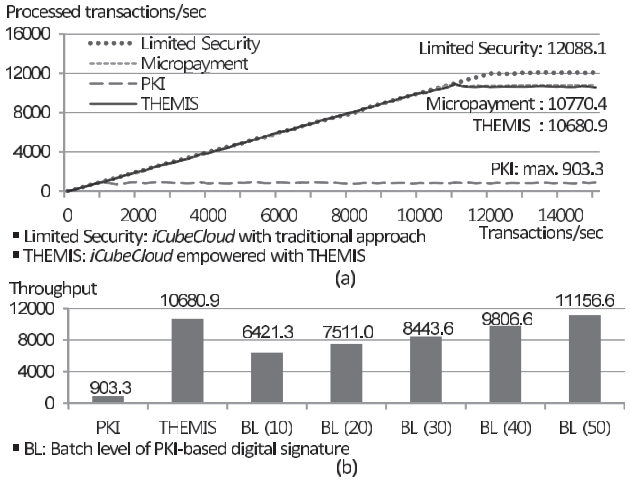


Fig. 11. (a) Throughput of the billing transactions with varying numbers of billing transactions per second; (b) throughput of THEMIS and PKI-based billing with varying the batch size.

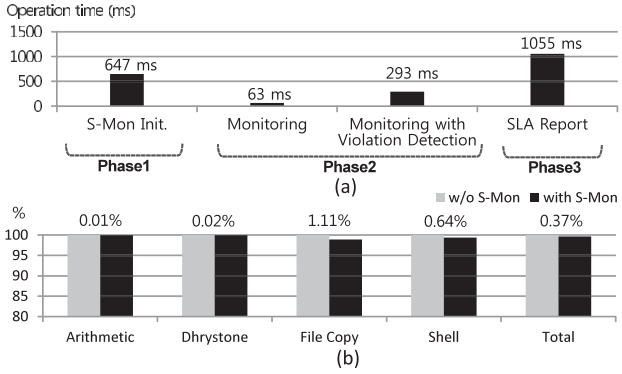


Fig. 12. Performance impact on users' cloud resource when S-Mon monitors SLA for user's service session.

second is less than 11,000. From the perspective of performance saturation, 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 toward a THEMIS-based system with more fault tolerance to scalable billing.

To investigate the tradeoff between performance and security, we measured the throughput of the PKI-based billing protocol while varying the batch size. For instance, both the cloud user and the CSP can digitally sign a sequence of transactions for every n transaction as opposed to digitally signing every transaction. Considering performance, maintaining a large batch size improves the throughput. Meanwhile, from a security perspective, achieving a digital signature per transaction prevents an advisory from modifying the service contract. Fig. 11b depicts the throughput for each billing scheme. By comparing the cost of THEMIS with the batch signature approach, we confirm that THEMIS, while retaining mutual verifiability at a per-transaction level, has a computation overhead level that is similar to that of the batch signature approach with a batch level of 50. This result arises because THEMIS drastically minimizes the billing transaction overhead.

5.4 Performance Impact of S-Mon

The goal of our next experiment is to examine the performance impact when S-Mon is applied to the cloud computing platform. To measure the S-Mon monitoring overhead, one cloud computing resource hosts four virtual machines which are assigned to users. In addition, each virtual machine is a 64-bit Linux 2.6.18 system with a 4-GB main memory. In the cloud computing platform, we use Xen 4.1 as a hypervisor. In this experiment, we measured the operating time of each phase (specifically, the initialization, monitoring, and report phases). Fig. 12a shows the operating time of each phase of S-Mon. The bar on the left-most side (Phase1) shows the time needed to initialize S-Mon. The operational overhead has an average execution time of 647 ms. The next two measurements (Phase2) show the operating time of the SLA monitoring phases. When monitoring without any type of SLA violation detection, the operational overhead averages 63 ms. When monitoring with SLA violation detection, the operational overhead averages 293 ms. The time difference is due to the overhead incurred when recording violations with the seal operation of S-Mon. The bar on the right-most side (Phase3) is measured at the end of the SLA monitoring session, and a report of the monitoring results is sent to the CNA. The operational overhead averages 1,055 ms.

The next experiment was done to examine the performance overhead of guest systems that are assigned to users when S-Mon is periodically invoked. The monitoring procedure of S-Mon was invoked every 3 seconds. The S-Mon driver on Xen gathers data on the system status for a period of 3 seconds, after which S-Mon reads the data from the driver. The performance overhead by S-Mon is measured using the UnixBench [37] benchmark. We measure the total execution time of the benchmark when S-Mon is applied to the cloud computing platform and compare it to the results of a system without S-Mon. The results of our experiment are shown in Fig. 12b. The black bars represent the normalized execution time when S-Mon is applied to the cloud computing platform in comparison with the system without S-Mon as the baseline (100 percent). The performance result implies from 0.01 to 1.11 percent performance overhead if S-Mon is applied to the cloud computing platform. This can be attributed to the very low overhead imposed by S-Mon, as it can achieve trusted SLA monitoring with system overhead of less than 1.11 percent.

6 Discussion and Further Works

In this section, we discuss additional issues, including an instance of actual deployment from the perspective of feasibility and additional extensibility, which may benefit from the security properties of THEMIS.

6.1 Real Deployment and Its Feasibility

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. The cost of these security features is that THEMIS requires additional roles by CSPs and the CNA. The CSP needs to deploy hardware resources equipped with TPM and TXT if CSP intends to provide additional assurances to their users. This is not a great impediment, as TPM and TXT are widely deployed security technologies in commercial off-the-shelf computer systems. CNA will play the additional role of a trusted third party, as discussed in Section 3.

Our next step is to consider the scalability and fault tolerance of THEMIS. Currently, we are investigating THEMIS 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 toward a THEMIS-based system with more fault tolerance against scalable billing.

6.2 Extensibility for Various Target Services

In this work, THEMIS is focused on SLA-monitoring, especially for IaaS services (in terms of availability and performance) and related billing transactions for mutual verifiability. From the perspective of extensibility, THEMIS should be naturally applicable to various target services as well to improve the accountability of each service. For

instance, by applying monitoring techniques such as [17], [18], [19] to S-Mon, we believe that THEMIS can facilitate the cloud-based services with accountability. Examples include PaaS, SaaS, and a cloud storage service. This type of facilitation is possible as long as the monitoring techniques can be plugged into the internal monitoring module of S-Mon. As a result, we believe that the complementarity of THEMIS and the existing monitoring techniques significantly improve the extensibility of this work.

6.3 Multi-CNA Support

If different users subscribe to different CNAs on a single physical resource, it becomes necessary for multiple S-Mons to be deployable for the multiple CNAs. Implementing this support requires multiple S-Mons to be invoked on a single physical resource. In addition, S-Mon needs to implement locking primitives to synchronize access to its global data of a physical TPM because multiple S-Mons share the physical TPM. By equipping each S-Mon with a virtual TPM (vTPM) device [38] involving the virtualization of the hardware TPM, S-Mon can overcome this restriction. The vTPM component proposes a method of virtualizing the hardware TPM. It provides the illusion of a physical TPM to S-Mons running on a single physical resource. This facility enables the multiple invocation of S-Mon on a physical TPM. Each S-Mon can be mapped to a different CNA because each S-Mon can have a different public key of CNA. Thus, the multiple CNA support would be a promising augmentation of this paper.

7 Conclusion

Our aim in this study was to provide a full-fledged trusted, nonobstructive 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.

References

[1] Amazon Web Services, "Amazon Elastic Compute Cloud EC2, Simple Storage Service," http://aws.amazon.com/ec2, http://aws.amazon.com/s32, Apr. 2011.

[2] Microsoft, "Microsoft, Windows Azure Platform," http://www.microsoft.com/windowsazure, 2010.

[3] M. Armbrust and A.E. Fox, "Above the Clouds: A Berkeley View of Cloud Computing," Technical Report UCB/EECS-2009-28, Electrical Engineering and Computer Sciences Dept., Univ. of California, Berkeley, Feb. 2009.

[4] N. Santos, K.P. Gummadi, and R. Rodrigues, "Towards Trusted Cloud Computing," *Proc. Conf. Hot Topics in Cloud Computing (HotCloud)*, 2009.

[5] R.T. Snodgrass, S.S. Yao, and C. Collberg, "Tamper Detection in Audit Logs," *Proc. 30th Int'l Conf. Very Large Data Bases (VLDB '04)*, pp. 504-515, 2004.

[6] L. Cornwall, M. Craig, R. Byrom, and R. Cordenonsib, "APEL: An Implementation of Grid Accounting Using R-GMA," *Proc. UK E-Science All Hands Conf.*, Sept. 2005.

[7] F. Tannenbaum, L. 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," *Proc. IEEE 13th Int'l Symp. High Performance Distributed Computing*, pp. 278-279, 2004.

[9] "Tivoli: Usage and Accounting Manager," IBM press release, 2009.

[10] PKIX Working Group, http://www.ietf.org/html.charters/pkix-charter.html, 2008.

[11] A. Guarise, R. Piro, and A. Werbrouck, "Datagrid Accounting System—Architecture—v1.0," technical report, EU DataGrid, 2003.

[12] P. Gardfill, E. Elmroth, L. Johson, O. Mulmo, and T. Sandholm, "Scalable Grid-Wide Capacity Allocation with the SweGrid Accounting System (SGAS)," *Concurrency Computation: Practice Experience*, 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," *Proc. 17th Int'l Symp. Parallel and Distributed Processing (IPDPS '03)*, pp. 22-26, 2003.

[14] G. von Voigt and W. Muller, "Comparison of Grid Accounting Concepts for D-Grid," *Proc. Cracow Grid Workshop*, pp. 459-466, Oct. 2006.

[15] NexR, "iCube Cloud Computing and Elastic-Storage Services," http://www.nexr.co.kr/, Mar. 2011.

[16] H. Rajan and M. Hosamani, "Tisa: Toward Trustworthy Services in a Service-Oriented Architecture," *IEEE Trans. Services Computing*, vol. 1, no. 4, pp. 201-213, Oct.-Dec. 2008.

[17] S. Meng, L. Liu, and T. Wang, "State Monitoring in Cloud Datacenters," *IEEE Trans. Knowledge and Data Eng.*, vol. 23, no. 9, pp. 1328-1344, Sept. 2011.

[18] C. Olston and B. Reed, "Inspector Gadget: A Framework for Custom Monitoring and Debugging of Distributed Dataflows," *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '11)*, pp. 1221-1224, 2011.

[19] P. Leitner, A. Michlmayr, F. Rosenberg, and S. Dustdar, "Monitoring, Prediction and Prevention of SLA Violations in Composite Services," *Proc. IEEE Int'l Conf. Web Services (ICWS)*, pp. 369-376, 2010.

[20] S. Pearson and B. Balacheff, *Trusted Computing Platforms: TCPA Technology in Context*. Prentice Hall Professional, 2003.

[21] "Intel Trusted Execution Technology, Hardware-Based Technology for Enhancing Server Platform Security," white paper, Intel, 2010.

[22] A. Haeberlen, "A Case for the Accountable Cloud," *SIGOPS Operating Systems Rev.*, vol. 44, pp. 52-57, Apr. 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," *Proc. IEEE Second Int'l Conf. Cloud Computing Technology and Science (CloudCom)*, pp. 687-692, 2010.

[24] R. Buyya, D. Abramson, J. Giddy, and H. Stockinger, "Economic Models for Resource Management and Scheduling in Grid Computing," *J. Concurrency Computation: Practice and Experience*, vol. 14, pp. 1507-1542, 2002.

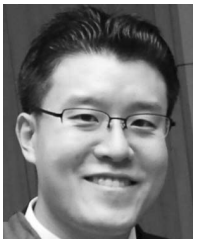
[25] B.N. Chun and D.E. Culler, "Market-Based Proportional Resource Sharing for Clusters," technical report, 1999.

[26] A. Herzberg and H. Yochai, "MiniPay: Charging per Click on the Web," *Proc. Selected Papers from the Sixth Int'l Conf. World Wide Web*, pp. 939-951, 1997.

[27] X. Dai and J. Grundy, "NetPay: An Off-Line, Decentralized Micro-Payment System for Thin-Client Applications," *Electronic Commerce Research Applications*, vol. 6, pp. 91-101, Jan. 2007.

[28] G.O. Karame, A. Francillon, and S. Čapkun, "Pay as You Browse: Microcomputations as Micropayments in Web-Based Services," *Proc. 20th Int'l Conf. World Wide Web (WWW)*, pp. 307-316, 2011.

[29] Y. Chen, R. Sion, and B. Carbunar, "XPay: Practical Anonymous Payments for tor Routing and Other Networked Services," *Proc. Eighth ACM Workshop Privacy in the Electronic Soc.*, pp. 41-50, 2009.
[30] Amazon Web Services, http://aws.amazon.com/cloudwatch/, 2010.
[31] R.L. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Comm. ACM*, vol. 26, pp. 96-99, Jan. 1983.
[32] K.-W. Park, S.K. Park, J. Han, and K.H. Park, "THEMIS: Towards Mutually Verifiable Billing Transactions in the Cloud Computing Environment," *Proc. IEEE Third Int'l Conf. Cloud Computing*, pp. 139-147, 2010.
[33] D. Challener, K. Yoder, and R. Catherman, *A Practical Guide to Trusted Computing*. IBM Press, 2008.
[34] "Trusted Boot: Open Source, Pre-Kernel/VMM Module," http://tboot.sourceforge.net/, 2011.
[35] J. Cihula, "Trusted Boot: Verifying the Xen Launch," Intel presentation at Xen Summit, Oct. 2007.
[36] C. Li, A. Raghunathan, and N.K. Jha, "Secure Virtual Machine Execution under an Untrusted Management OS," *Proc. IEEE Int'l Conf. Cloud Computing*, pp. 172-179, 2010.
[37] The UNIX and Linux Forums, "UNIX Benchmarks," http://www.unix.com/unix-benchmarks/, 1991.
[38] S. Berger, R. Cáceres, K.A. Goldman, R. Perez, R. Sailer, and L. van Doorn, "vTPM: Virtualizing the Trusted Platform Module," *Proc. 15th Conf. USENIX Security Symp.*, vol. 15, 2006.



Ki-Woong Park received the BS degree in computer science from Yonsei University in 2005 and the MS and PhD degrees in electrical engineering from the Korea Advanced Institute of Science and Technology in 2007 and 2012, respectively. He is currently an assistant professor in the Computer Hacking and Information Security Department at Daejeon University. He worked as a researcher at the National Security Research Institute in 2012. His research interests include security issues for cloud and mobile computing systems as well as the actual system implementation and subsequent evaluation in a real computing system. He received a 2009-2010 Microsoft Graduate Research Fellowship. He is a member of the IEEE and the ACM.

Jaesun Han received the BS degree in electrical engineering from Pusan National University in 1998 and the MS and PhD degrees in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST) in 2000 and 2005, respectively. He is the founder and CEO of NexR, which has been developing a big data analytics platform and a cloud computing platform since 2007. He has been an adjunct professor at KAIST Business School since 2007. He founded and has led the Korean Hadoop User Group since 2009. His research interests include big data analytics and cloud computing. He is a member of the IEEE.

JaeWoong Chung received the BS and MS degrees from the Korea Advanced Institute of Science and Technology in 1997 and 1999, respectively, and the PhD degree from Stanford University in 2008. He is a research scientist at Intel Labs. He was at Advanced Micro Devices from 2008 to 2010, TmaxSoft from 2001 to 2004, and Samsung from 1999 to 2000. His research interests include enterprize system design, transactional memory, and cloud computing. He is a member of the IEEE.

Kyu Ho Park received the BS degree in electrical engineering from Seoul National University, Korea, 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 de Paris XI, France, in 1983. He has been a professor in the Division of Electrical Engineering at KAIST since 1983. His research interests include computer architectures, storage systems, cloud computing, and parallel processing. He is a member of the IEEE and the ACM.