DIaaS: Data Integrity as a Service in the Cloud

Conference Paper · July 2011

DOI: 10.1109/CLOUD.2011.35 · Source: DBLP

4 authors:

Surya Nepal
The Commonwealth Scientific and Industrial …
192 PUBLICATIONS **1,502** CITATIONS
SEE PROFILE

Shiping Chen
The Commonwealth Scientific and Industrial …
126 PUBLICATIONS **764** CITATIONS
SEE PROFILE

Kim Jinhui Yao
Xerox Corporation
28 PUBLICATIONS **134** CITATIONS
SEE PROFILE

Danan Thilakanathan
University of Sydney
11 PUBLICATIONS **68** CITATIONS
SEE PROFILE

2011 IEEE 4th International Conference on Cloud Computing

DIaaS: Data Integrity as a Service in the Cloud

Surya Nepal, Shiping Chen, Jinhui Yao, Danan Thilakanathan
Information Engineering Lab
CSIRO ICT Centre
Sydney, Australia
Firstname.lastname@csiro.au

***Abstract*— In this paper, we propose a secure cloud storage service architecture with the focus on *Data Integrity as a Service* (DIaaS) based on the principles of Service-Oriented Architecture and Web services. Our approach not only releases the burdens of data integrity management from a storage service by handling it through an independent third party data Integrity Management Service (IMS), but also reduces the security risk of the data stored in the storage services by checking the data integrity with the help of IMS. We define data integrity protocols for a number of different scenarios, and demonstrate the feasibility of the proposed architecture, service and protocols by implementing them on a public cloud, Amazon S3. We also study the impact of our proposed protocols on the performance of the storage service and show that the benefits of our approach outweigh the little penalty on the storage service performance.**

Keywords- cloud storage; data integrity; security; data integrity as a service.

I. INTRODUCTION

Cloud computing is touted as the next generation of computing and considered to make significant advances in the next decade. The driving forces behind cloud computing are: (a) significantly reduced *Total Cost of Ownership* (TCO) of the required IT infrastructure and software including (but not limited to) purchasing, operating, maintaining and updating costs; (b) high *Quality of Service* (QoS) provided by cloud service providers such as availability, reliability and Pay-As-You-Go (PAYG) based low prices; (c) easy access to organizational information and services *anytime anywhere*; and (d) dynamic resource scaling based *on demand*. In a nutshell, cloud computing provides a new paradigm for delivering computing resources to customers on demand in a similar fashion as utilities [1].

The current cloud computing architecture enables clients to interact with servers by providing three layers of services [2]: *Software as a Service* (SaaS) provisions complete applications as a service, such as Customer Relationship Management (CRM). *Platform as a Service* (PaaS) provides a platform for developing other applications on top of it, such as the Google App Engine (GAE). Finally, *Infrastructure as a Service* (IaaS) provides an environment for deploying, running and managing virtual machines and storage. Technically, IaaS provides incremental scalability (scale up and down) of computing resources and nearly unlimited, on-demand storage. Our focus in this paper is on *cloud storage service*.

Cloud storage services have emerged as a way to address the effective utilization of storage space to meet explosive growth of personal and enterprise data. They allow clients to scale their storage space requirements to meet expanding needs while improving utilization and manageability. For example, the storage clouds such as Amazon S3 , Google documents, and RackSpace demonstrate the feasibility of storage services in a new computing paradigm by offering (almost) unlimited storage for free or at very low prices yet with high availability (24X7 days). Given its innovative nature compared to the standard model of service provision, cloud computing raises new questions in terms of security [3]. There is a need to understand the risks associated with it as well as to build technologies to address those risks.

Cloud storage services are not secure by nature. The security challenges related to cloud storage services need deeper attention. In terms of security, cloud storage services must be managed and operated at equivalent security levels to enterprise storage systems. However, cloud users typically have no control over the cloud storage servers used. This means there is an inherent risk of data exposure to third parties on the cloud or by the cloud provider itself (*data confidentiality*); to data tampering by the third party on the cloud or by the cloud provider itself (*data integrity*); and to denial of data by third parties on the cloud or by the cloud provider itself (*data availability*). The cloud storage should ensure data *confidentiality, integrity* and *availability* (CIA) both in motion (while transmitting over networks) and at rest (when storing at providers' disks). The issues of CIA in cloud storage service have been studied in literature [4][5][6]. We observed that confidentiality has received a greater attention from the researchers in recent times [7][8][9][10][32]. We have also addressed the issue of *data confidentiality* (C) in our earlier work [11] by proposing a cloud storage service architecture and corresponding data encryption algorithms and protocols with the help of a key management service. Though the issues of data integrity and availability are equally important, there has been less focus on these issues. This paper addresses the issue of *data integrity* (I).

The data integrity problem in the cloud has been studied from a variety of perspectives ranging from storage technologies [14][16][18][20][22] to architecture [24][25] to standards [28][30] (see details in Section II). One of the drawbacks of these approaches is that the data integrity problem has been coupled together with other problems and has not received treatment in isolation. As a result, the

978-0-7695-4460-1/11 $26.00 © 2011 IEEE
DOI 10.1109/CLOUD.2011.35

proposed data integrity mechanisms rely on the architectural components dedicated for other tasks such as data storage, certifying authority, key management service and transaction monitor. We deviate from this approach and treat the data integrity problem in itself by proposing the *Data Integrity as a Service* (DIaaS, *pronounced as "Diaz"*) in the cloud architecture. This enables to gather all the expertise related to data integrity in one place so that it can deal with some of the well-known problems in this domain such as public verifiability and dynamic content [27]. The key contributions of this approach paper can be summarized as follows.

- The paper treats the data integrity management in the cloud in isolation by introducing a concept DIaaS into the cloud architecture based on the principles of SOA and Web services. As a result, the trust on cloud storage services is enhanced by independently checking customers' data integrity using the 3rd party DIaaS.
- The paper proposes protocols for handling data integrity violations through a data integrity service under four different circumstances: (a) trusted data integrity service, (b) un-trusted data integrity service, (c) dependent on the underlying storage service provider, and (d) independent of the storage service provider.
- The paper demonstrates the feasibility of the proposed concept, architecture and protocols by implementing a proof-of-concept demonstrator on a public cloud, Amazon S3, by composing data integrity service with other services using the concept of service composition.

The rest of the article is structured as follows. Section II presents the related work with an aim of providing the motivation by reviewing the existing literature. Section III describes the secure cloud storage architecture and how it treats data integrity as a service. The data integrity service could be implemented within a trusted environment or un-trusted environment. Similarly, a storage service provider may not support functionalities needed for data integrity that is compatible with the client environment. Section IV presents the data integrity protocols for these different possible scenarios. We have implemented the architecture, service and protocols in the public cloud, Amazon S3 and conducted a performance analysis, which is described in Section V. The last section draws the concluding remarks and potential future work.

II. Related Work

Data integrity is a fundamental aspect of cloud storage security [4][12]. As data is at the heart of cloud storage service, it is important that the cloud service providers provide assurance of data integrity to its users. This is one of the areas in cloud storage security that is often overlooked. Too often, it is assumed that underlying storage arrays receive, store and retrieve data flawlessly. This assumption is is proven to be not true in recent times, as evident from the CERN report [13]. Therefore, prompt detection of integrity violations is vital for the reliability and safety of the stored data in the cloud. There are many technologies and protocols that have been proposed to address the issue of data integrity in the cloud environment. We review them under the following four categories.

1) Storage Server Technologies

Sivathanu, Wright, and Zadok [14] have presented a review of data integrity mechanisms in generic storage file systems. They are applicable to cloud storage services as well. There are mainly three most common integrity assurance techniques: Mirroring [15], RAID [16] and Checksumming [17][18][19]. The mirroring is a very simple way of implementing integrity verification of data by maintaining two or more copies of the same data in the storage space [15]. Integrity checks can be made by comparing the copies. Parity is used in RAID-3, RAID-4, and RAID-5 [16] to validate the data written to the RAID array. HAIL is an example cloud storage architecture based on this technology [20]. While RAID manages sector redundancy dynamically across hard-drives, HAIL manages file redundancy across cloud storage providers. Similarly, Zetta has proposed a new RAID for cloud-scale data integrity, called RAIN [21]. One of the most popular methods used to ensure integrity of the data in transit over the public un-trusted internet is checksumming. Checksums are computed using a cryptographic hash function such as MD5 [19], SHA1 [17] and HMAC [18]. Data integrity can be verified by comparing the stored and newly computed hash results. Our proposed protocols use this concept.

2) Protocols

A number of algorithms and protocols have been proposed in the literature to address the problem of secure verification that an un-trusted outsourced storage service is faithfully storing the client data [22][23][24][25][26][27]. They are generally studied under two categories: *Provable Data Possession* (PDP) and *Proof of Retrievability* (PoR). Ateniese et al. [22] define the PDP model that enables un-trusted servers to verify the possession of the original data without retrieving it. Their protocol is defined for static file and can not be extended to dynamic data storage without introducing security loopholes easily. This problem has been addressed in their follow up work in [23]. However, their solution does not support fully dynamic data operations. Wang et al. [24] proposed a challenge-response protocol, but they also consider partial support for dynamic data operations.

Juels and Kaliski [25] describe a PoR model that enables a user to determine that a server possesses a file. A POR permits detection of tampering or deletion of a remotely located file or relegation of the file to storage with uncertain service quality. Furthermore, this scheme suffers from the lack of support for dynamic operations. Erway et al. [26] are the first to explore constructions for dynamic provable data possession. They extend the PDP model in [22] to support provable updates to stored data files. However, the efficiency of their scheme remains in question. All these schemes can be implemented in our proposed data integrity service. However, their reliance on storage service still remains a problem.

3) Standards

There have been some approaches to address the data integrity problem in the cloud through standards. In the database management systems, "ACID" properties are used to support data integrity [29]. The question is whether a particular cloud computing solution offers similar mechanism [34]. Over time, more cloud service providers may refer to developing standards such as the SNIA Cloud Data Management Interface (CDMI) specification [30] and other SNIA cloud storage standards. These approaches would help the customer feel more confident that good data gets into cloud databases, stays there, and comes out of the cloud in the same shape.

4) Storage Architecture

The data integrity solutions for the clouds using storage technologies and client-server protocols have largely ignored the need of the public verifiability of the data for the purpose of accountability and auditing. To address this concern, researchers have looked for solutions at the architecture level, mainly by introducing a third party verifier [24]. Our proposed concept *DIaaS* falls in this category. Wang et al. [24] have introduced a concept of third party auditor on behalf of the cloud client to verify the integrity of the dynamic data stored in the cloud. The paper addresses two major issues with PDP and PoR models: public verifiability and support dynamic operations on stored data. This approach is vulnerable towards user data privacy. To address this fear, the approach is extended in [27] by proposing a privacy-preserving protocol.

Along with other approaches, this approach is very similar to our earlier approach proposed in [11], where the key management service in conjunction with cloud storage service handles the issue of data integrity. In this paper, we deviate from these earlier approaches, where the burden of checking integrity violations is dealt by one of the architectural components defined for other purposes, by proposing a dedicated service for data integrity, called *Integrity Management Service* (IMS).

III. TRUSTED CLOUD STORAGE ARCHITECTURE

We have presented a secure cloud storage architecture, called TrustStore, in [11]. In this paper, we extended the architecture to include the concept of DIaaS through a data integrity service. The extended architecture is shown in **Error! Reference source not found.**The architecture contains five key components: *Client Application Portal* (CAP), *Trust Management Service* (TMS), *Cloud Storage Service (CSS)*, *Key Management Service* (KMS) and *Integrity Management Service* (IMS). We briefly describe each component of the architecture as the following.

Client Application Portal (CAP): The client application resides in the trusted environment. The client application has two parts. The front part is the GUI with which the client interact the backend of CAP. The GUI supports the basic functionality of storing, retrieving and browsing data stored in the cloud. The backend of the CAP includes a driver that implements the algorithms and protocols. For example, the driver segments a data file into a random number of segments stored in the cloud storage service. In this scenario, the driver deals with algorithms and protocol related to *confidentiality* of the data, which includes encryption and decryption of data. In summary, the backend of the client application implements all core essential protocols and processing algorithms in our architecture. One can implement it as a downloadable driver.

Trust Management Service (TMS): Not all cloud storage services are created identically. If the wrong provider is chosen, all the benefits can vanish. The question is then how can you ensure the best possible storage services are selected for you and your business?. There are increasing numbers of solutions available. Not all of them are capable of or keen on providing adequate level of Quality of Service (QoS) and deliver the promised qualities. When storing encrypted and segregated data, the data must be stored in a service that meets the required QoS. Therefore, *trust* becomes an important and essential issue in selecting the appropriate cloud storage services, where *trust* is defined as the reflection of the QoS provided by the cloud storage services. The trust management service provides a reputation based trust management system for cloud storage service providers.

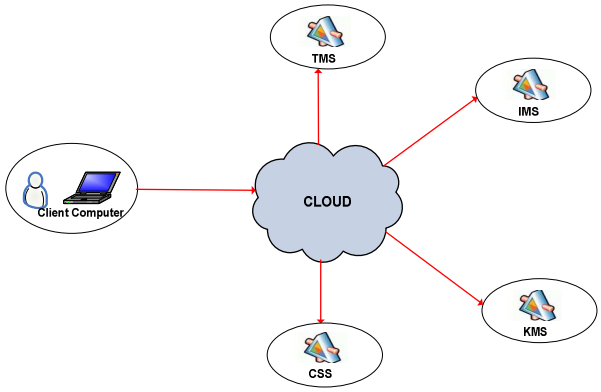

Figure 1. Secure loud Storage with Integrity Management as a Service.

Key Management Service (KMS): One of the most common, and yet significant problems in providing secure storage service in the cloud is the key management (i.e., generation and maintenance of cryptographic keys described earlier). In our architecture, we utilize a dedicated key manager known as the Ephemerizer [31] to play this role. This service enables users to 'keep' data for a finite time and then makes them unrecoverable after that, which can be one of the desirable policies related to data to maintain privacy. The Ephemerizer service creates keys, makes them available for encryption, aids in decryption, and destroys the keys when they expire. We can make the data persistent by defining the expiration time for the keys infinity. This service also keeps a database of the mapping information that links data with ephemeral keys. The service may routinely obtain a list of expired ephemeral keys which will be translated into corresponding data objects. Depending on the policy these objects can be used to instruct the cloud storage service to reclaim storage space occupied by expired (disappeared) data.

Cloud Storage Service (CSS): There are many cloud storage service providers providing storage services in the cloud. For example, Amazon S3, Google documents, and RackSpace demonstrate the feasibility of storage services in the public cloud. Many enterprises deploy private cloud storage for the internal use (also referred to as data center) [33] . Our focus in this article is on the public clouds.

Integrity Management Service (IMS): The integrity management service is a core component of implementing the concept *DIaaS* and deals with the integrity of the data stored in the storage service. A cloud storage service may tamper the data while at rest. Similarly, a third party could tamper the data in transit. Furthermore, a salient corruption may occur in the data [13]. Nevertheless, the public cloud storage service is inherently not trusted. In order to address this problem, we introduce the concept of *DIaaS* and define a third party service in the cloud architecture to deal with the problem of integrity violations of the cloud data.

The general working of the architecture can be explained as follows. The client specifies the confidential data to be uploaded to the cloud with the client application. The confidential data is then processed with an encryption mechanism to transform the data into two parts: cipher text and encryption key. The client application then finds a suitable CSS using TMS that meets the client's QoS requirements using a reputation-based trust provided by TMS. The cipher text is uploaded to CSS. Without the key, the CSS can not reverse the process and access to the confidential data. The key is stored with KMS. It has the capability to reverse the encryption process, but it does not have the cipher text. The client application also calculates the hash value of the stored data and stores the hash value in the IMS. The CSS can change the data, but it will be detected by the IMS. The hash value stored with IMS is signed by both client application and IMS so that its authenticity can be established later.

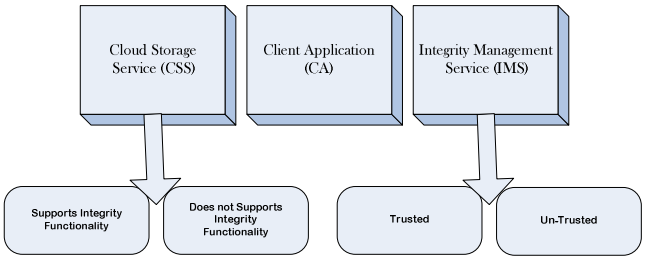

Figure 2. Key Components Integrity Management as a Service.

IV. DATA INTEGRITY AS A SERVICE (DIAAS)

This section describes the models and protocols for the Integrity Management Service (IMA), architectural component realizing DIaaS.. The key components of the architecture involves in the data integrity management is shown in **Error! Reference source not found.** As can be seen in the figure, there are four different scenarios. We define data integrity protocols for combinations of each of these scenarios as follows. Before describing the protocols, we describe the notations used in the protocols in TABLE I.

TABLE I. NOTATION USED FOR DESCRIBING PROTOCOLS

Notation	Description
CA	*: Client Application*
CSS	*: Cloud Storage Service*
IMS	*: Integrity Management Service*
Hash	*: The function that calculates the hash value*
Dataobject	*: Data to be stored in CSS*
ObjectID	*: Data reference in CSS*
$Sign_c$	*: Signed by component C's key*
$Verify_c$	*: Verified by component C's key*
C_{pri}	*: Private part of long term key of "C"*
C_{pub}	*: Public part of long term key of "C"*
Comp	*: Function to compare two hash values*
Enc_c	*: Encrypted by component C's public key*
Dec_c	*: Decrypted by component C's prkey*
Store	*: Storing the content*
Retrieve	*: Retrieving the relevant content*

A. Model CSS-Y IMS-U

This model (CSS-Y IMS-U) refers to the situation where the cloud storage service supports cryptographic functionalities for the data integrity such as the return of a hash value of the stored data in the cloud, but the client does not trust the data integrity management service. Cloud storage services such as Amazon S3 return MD5 hash of the stored data.. We define the protocols for uploading and downloading data files in this model. The uploading protocol is shown in Figure 3. We explain the protocol further below.

1.	CA	: Hash(DataObject)
2.	CA → CSS	: DataObject
3.	CSS→ CA	: (ObjectID, objecthash)
4.	CA	: Comp(objecthash, Hash(Dataobject))
5.	CA → IMS	: ObjectID
6.	IMS → CSS	: ObjectID
7.	CSS→ IMS	: objecthash
8.	IMS	: $Sign_{CA}$(objecthash, ObjectID, time)
9.	IMS →CA	: $Sign_{CA}$(objecthash, ObjectID, time)
10.	CA	: $Verify_{CA}$(objecthash, ObjectID, time) : Comp(objecthash, Hash(Dataobject)) : $Sign_{IMS}$($Sign_{CA}$(objecthash, ObjectID, time) : Enc_{IMS}(NONCE)
11.	CA →IMS	: $Sign_{CA}$($Sign_{IMS}$(objecthash, ObjectID, time)) : Enc_{IMS}(NONCE)
12.	IMS	: $Verify_{CA}$($Sign_{CA}$($Sign_{IMS}$(objecthash, ObjectID, time))) : Dec_{IMS}(NONCE) : Enc_{CA}(NONCE) : Store($Sign_{CA}$($Sign_{IMS}$(objecthash, ObjectID, time)))
13.	IMS → CA	: Enc_{CA}(NONCE)
14.	CA	: Enc_{CA}(NONCE)

Figure 3. Data uploading protocol for Model CSS-Y IMS-U

It is assumed that the CA has a cryptography library supporting different types of checksumming functionalities such as MD5, SHA-1 and HMAC. The CA first applies a hash function, creates a unique hash value of the object and sends it to the CSS (steps 1 & 2). The CSS stores the data object and returns the hash value of the stored object (step 3). The CA then compares two hash values of the same data object (step 4). If the hash value matches, the CA sends an object id to the IMS (strep 5). The IMS uses this object id to retrieve the hash value from CSS, signs it with its private key and sends the signed hash value back to the CA (steps 6, 7, 8 and 9). The CA compares the hash value in IMS with its

original hash value, verifies the signature to ensure that the hash value was sent by IMS and then signs it using the private key and sends back to the IMS (steps 10 and 11). These steps also include, creating a NONCE, encrypting it with the public key of IMS, and sending it to the IMS, to ensure that IMS has really stored the hash value of the data object. The IMS then verifies the signature, decrypts the NONCE, stores the hash value, and encrypts the NONCE by the public key of CA and sends back to the CA (steps 12 and 13). The CA then decrypts the NONCE and compares the value with the original NONCE (step 13). This step ensures that the correct hash value is stored in the IMS and the signed NONCE can be used later to ensure this.

The protocol for downloading data from the cloud in the original form by checking the possible integrity violations is shown in Figure 4. The CA first sends the object id to the CSS and retrieves the data object. The hash value is calculated on the data object. The CA then sends a request to the IMS to retrieve double signed hash value of the original data object stored during the uploading process described earlier. The CA verifies the signatures and compares the hash value of the original file (retrieved from the IMS) with the hash value calculated from the file obtained from CSS. The successful comparison indicates that no violations of data integrity occur on the data (both in transit and at rest).

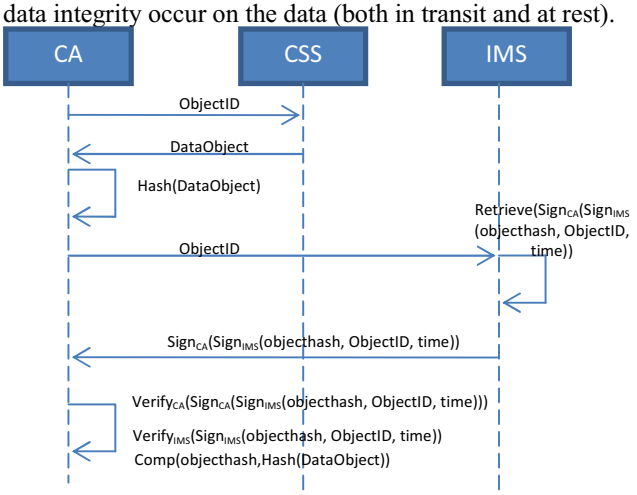

Figure 4. Data downloading protocol for CSS-Y, IMS-U.

B. *Model CSS-Y IMS-T*

This model (CSS-Y IMS-T) refers to the situation where the cloud storage service supports functionalities for data integrity such as the return of a hash value of the stored object in the cloud and the data integrity service is trusted. In this model, the data integrity service is either provided by the client itself or a trusted third party. This is a more specific model as it imposes the client trusting both IMS and CSS supporting the required integrity functionalities. The data uploading and downloading protocols for this model is shown in Figure 5. Due to the trusted nature of the IMS, the protocols in this model are simple. The CA gets the hash value of the data object and sends it to IMS. There is no need to do independent verifications of the hash value.

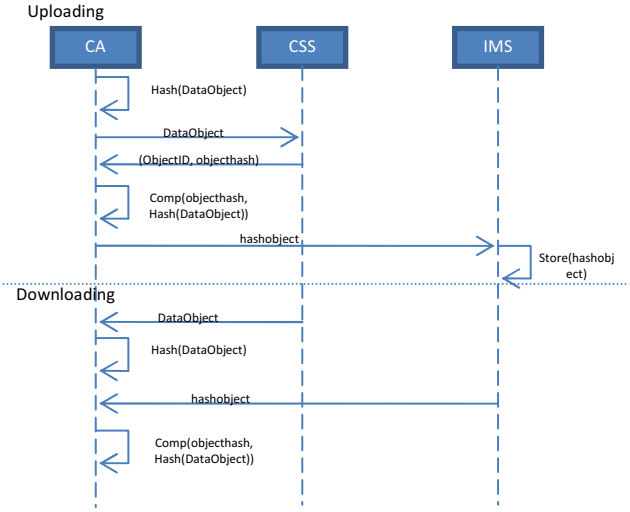

Figure 5. Data uploading and downloadong protocols for model CSS-Y IMS-T.

C. *Model CSS–N IMS-T*

This model (CSS-N IMS-T) refers to the situation where the cloud storage service does not support the functionalities for data integrity such as the return of the hash value of the stored data object in the cloud and the data integrity service is trusted. This model is independent of the underlying cloud storage service.

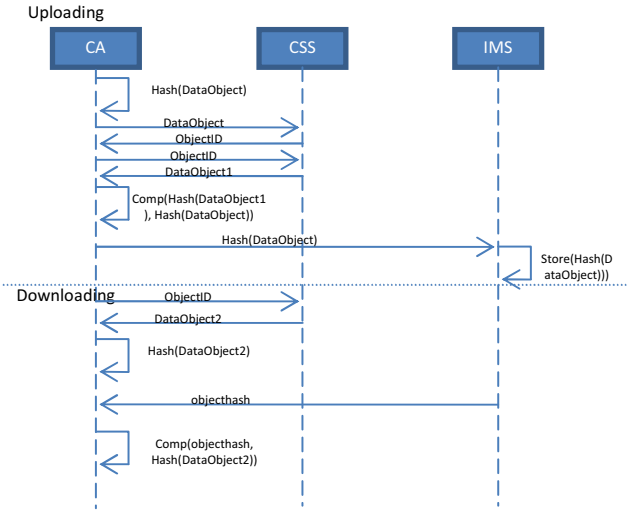

Figure 6. Downloading protocol for CSS-Y, IMS-U.

The data uploading and downloading protocols for this model is shown in Figure 6. As this model work independent of the underlying CSS, the model needs to take extra care while uploading the data object as compare to the model with CSS-Y. The CA calculates a hash value of the original data object and uploads to CSS. The CSS then returns the object id. In order to ensure that there are no violations to data integrity while transmitting and storing the data, the CA immediately read the data object from the CSS. The CA then

computes and compares the hash value of the original data object with the data object just read from CSS. If both hash values match, it ensures that the CSS stores the data object in the original shape and form. This process helps CA to ensure that the original data is stored in the cloud. The downloading protocol is similar to the model CSS-Y IMS-T.

D. Model CSS-N IMS-U

This model (CSS-N IMS-U) refers to the situation where the cloud storage service does not support functionalities for data integrity such as the return of a hash value of the stored object in the cloud and the data integrity service is un-trusted. This model is the most generic model in comparison to previous three models. This model does not make any assumptions on trustworthiness of both CSS and IMS.

1.	CA	: Hash(DataObject)
2.	CA → CSS	: DataObject
3.	CSS→ CA	: (ObjectID)
4.	CA→ CSS	: (ObjectID)
5.	CSS→ CA	: DataObject1
6.	CA	: Comp(Hash(DataObject1), Hash(Dataobject))
7.	CA → IMS	: ObjectID
8.	IMS → CSS	: ObjectID
9.	CSS→ IMS	: DataObject2
10.	IMS	: Hash(DataObject2)
11.	IMS	: $Sign_{CA}$(Hash(DataObject2), ObjectID, time)
12.	IMS →CA	: $Sign_{CA}$(Hash(DataObject2), ObjectID, time)
13.	CA	: $Verify_{CA}$(Hash(DataObject2), ObjectID, time) : Comp(Hash(DataObject2), Hash(Dataobject)) : $Sign_{IMS}$($Sign_{CA}$(Hash(DataObject2), ObjectID, time) : Enc_{IMS}(NONCE)
14.	CA →IMS	: $Sign_{CA}$($Sign_{IMS}$(Hash(DataObject2), ObjectID, time)) : Enc_{IMS}(NONCE)
15.	IMS	: $Verify_{CA}$($Sign_{CA}$($Sign_{IMS}$(Hash(DataObject2), ObjectID, time))) : Dec_{IMS}(NONCE) : Enc_{CA}(NONCE) : Store($Sign_{CA}$($Sign_{IMS}$(Hash(DataObject2), ObjectID, time)))
16.	IMS → CA	: Enc_{CA}(NONCE)
17.	CA	: Enc_{CA}(NONCE)

Figure 7. Data uploading protocol for the model CSS-N IMS-U.

The data uploading protocols for this model is shown in Figure 7. This model is very useful in the scenario where multiple cloud providers are used by a single CA.

V. PROTOTYPE SYSTEM

In order to illustrate our architecture, models and corresponding protocols, we have implemented a prototype system. We next describe the implementation of the prototype system and the results on the performance evaluation of the models.

A. Implementation

We have implemented the cloud storage service architecture. We have used Amazon S3 as CSS. Amazon S3 allows standard storage operations such as: PUT, GET, LIST, and DELETE. The client application directly executes these operations through sending SOAP messages to Amazon S3. KMS, IMS and TMS were implemented and deployed in local the PC as Web services. A client application is implemented as a GUI-based java application for users to interact with CSS and other services. We have implemented a client application for providing archive storage services for user files. At the core of the client application, we have a driver that takes care of tasks of data processing as well coordinating different services. This is done by implementing all the models and supporting protocols. Users can utilize the application to archive data file by a simple standard operation such as drag and drop. We refer readers to our earlier paper [11] on data confidentiality to obtain further details on these operations.

B. Performance Evaluation

This subsection presents the performance evaluation of our protocols for different models. We have implemented four different trust models in our prototype system. The report on our evaluation focuses on two aspects of the proposed IMS: (a) the cost of IMS on cloud storage services, and (b) the comparison of different models. We next describe the evaluation environments and results on those two aspects of our proposal.

To demonstrate the cost related to the implementing IMS protocols, we have conducted a baseline testing to observe the latency of our protocols for uploading and downloading files as compared to the ordinary use of Amazon S3. As described in previous section, the PUT/GET operations in our system involve several extra processes apart from the transmission, including hashing and signing as explained in the earlier section. We first show the testing results on each of these processes for both PUT and GET operations, and then compare different protocols.

The tests are carried out on a computer with an Intel(R) Core(TM) 2 CPU 6400 @2.13GHz and 1.99GB RAM. The IMS and client application were running on the same machine in the LAN environment, while the Amazon S3 server is likely located in the USA. We have used MD5 hash returned by Amazon S3 for CSS-Y and SHA-1 for CSS-N.

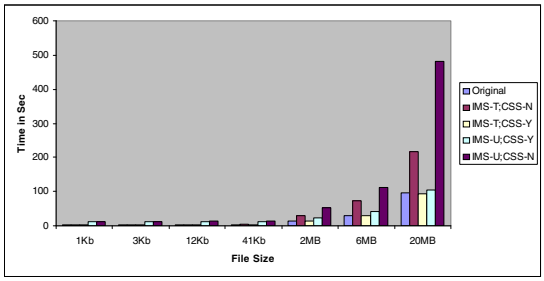

Figure 8. Performance of Uploading Different Files Under Different Models.

The performance results for uploading and downloading files for different models is shown in Figures 9 and 10, respectively. The result is the average over five iterations. We can derive the following observations from the results.

1. The generic model costs higher than other models. For example, CSS-N IMS-U model costs higher than other models in the above figures.

2. The comparative differences on the cost of downloading files for a generic model and other models are not that much. This means the cost of a generic model is not much for the files with the property "write once read many".
3. The cost of uploading and downloading is proportional to the size of the files.
4. The cost of integrity management is very low for the cloud storage services that support hash functionality.
5. The difference of costs on models with IMS trusted and un-trusted are not much. This is due to the locally deployed IMS. We expect the cost be higher if the IMS is deployed on the cloud (e.g., Amazon EC2).

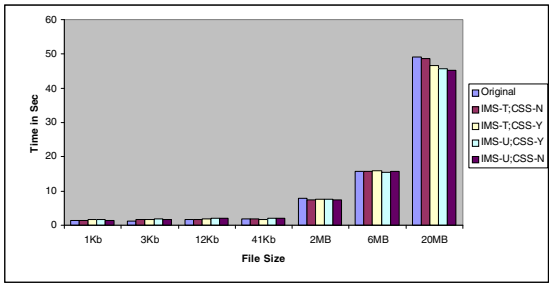

Figure 9. Performance of Downloading Different Files.

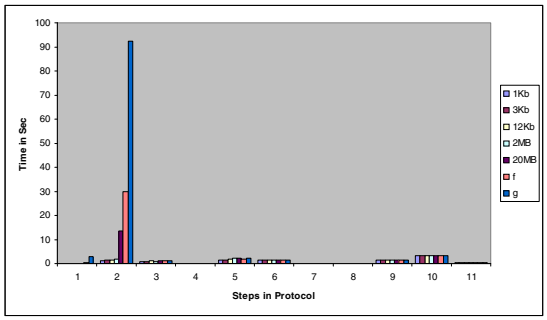

Figure 10. Cost of protocol steps in CSS-Y IMS-U Model.

The comparative cost differences on uploading and downloading files (observations 1 and 2 above) needs further investigation. We further investigated the cost of individual steps within the protocol. The cost of individual steps for the model CSS-Y IMS-U is shown in **Error! Reference source not found.** The steps are: (1) Start upload and compute MD5 (2) Upload file to AS3 (3) Get eTag/Compare eTag and MD5 hash (4) Notify upload to IMS (5) IMS get eTag from AS3 (6) IMS Sign eTag in XML format (7) Client verify signature (8) Client compare MD5 hash and eTag (9) Client double sign hash (10) Encrypt/Decrypt NONCE and (11) Store double sign hash in IMS. As you can see, the major cost is the transmission of the file to AS3. Therefore, the cost of the data integrity service depends on the transmission cost of the protocol messages between the client application and IMS.

We also compared the cost of most expensive (generic) model CSS-N IMS-U and the least expensive (specific) model CSS-Y IMS-T. The results are shown in Figure 11. As seen in the figure, the cost of a specific model is the same as standard operations for uploading and downloading files supported by underlying CSS (in our case AS3 PUT and GET operations). However, the cost of the generic model increases exponentially with the increase in the file size. Therefore, we need to pay a special attention while loading large files using the generic model.

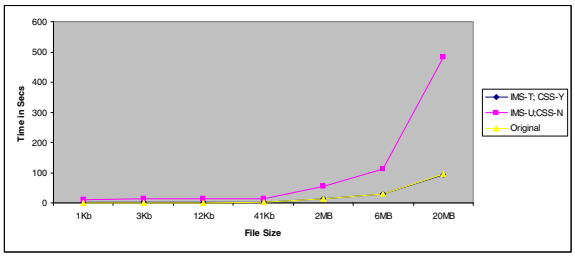

Figure 11. Cost Comparision of Expensive and Cheap Model with The Original.

VI. CONCLUSIONS

There are three key aspects of cloud data security: *confidentiality*, *integrity* and *availability* (also known as CIA). In this paper, we introduced a concept of *Data Integrity as a Service* (*DIaaS*, pronounced as "Diaz") to address the data integrity concerns in the cloud storage service. The major shortcoming of the existing techniques is that they rely on other components of the storage service architecture to perform this duty. As a result, they are weak at dealing with dynamic operations on data and the support of public verification. In this paper, we argued that these issues can be addressed better if we put all expertise relevant to data integrity at one place. This is achieved through the concept of *Data Integrity as a Service*.

We have extended the cloud storage service architecture by composing cloud storage service with data integrity service along with other services such as trust management service, key management service, etc. We considered the following two scenarios: (a) the cloud storage service (does not) support hash functionality, and (b) the data integrity service is (un)-trusted. We defined four different models to cover these scenarios and proposed corresponding protocols. We have implemented the data Integrity Management Service (IMS) as a Web service to realize *DIaaS* concept and composed it with public cloud storage services. We have conducted the tests and studied the impact of the protocols on the overall performance of the storage service. We have also made a number of interesting observations.

We are building a secure cloud storage service using the principle of Service-Oriented Architecture (SOA) and Web services, more specifically using the service

composition. We have dealt the issue of *confidentiality* through a *key management service* and reported the results in [11]. In this paper, we have dealt the issue of *integrity* through an *integrity management service*. In future, we plan to address the issue of *availability*. Another important issue in cloud computing is the interoperability between different clouds, as many public and private clouds have emerged in recent times without any mutually agreed standards. *DIaaS* is a suitable concept to deal with the data integrity issue across such diverse multiple clouds. We plan to validate this argument as one of our future works.

REFERENCES

[1] R. Buyya, C.S. Yeo, S. Venugopal, J. Broberg, and I. Brandic. "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility". *Future Gener. Comput. Syst.* vol. 25, no. 6, June 2009, pp. 599-616.

[2] A. Lenk, M. Klems, J. Nimis, S. Tai, and T. Sandholm. "What's inside the Cloud? An architectural map of the Cloud landscape". *Proc. Software Engineering Challenges of Cloud Computing*, 2009, pp. 23–31.

[3] G. Peterson. "Don't Trust. And Verify: A Security Architecture Stack for the Cloud". *IEEE trans. Security and Privacy* vol. 8, no. 5, 2010, pp. 83-86.

[4] L.M. Kaufman. "Data Security in the World of Cloud Computing". *IEEE Security and Privacy* vol. 7, no. 4, July 2009, pp.61-64.

[5] M. Jensen, J. Schwenk, N. Gruschka, and L.L. Iacono. "On Technical Security Issues in Cloud Computing". *Proceedings of the 2009 IEEE International Conference on Cloud Computing* (CLOUD '09). pp. 109-116.

[6] K. Hwang, S. Kulkareni, and Y. Hu. "Cloud Security with Virtualized Defense and Reputation-Based Trust Mangement". *Proceedings of the 2009 Eighth IEEE International Conference on Dependable, Autonomic and Secure Computing* (DASC '09). pp.717-722.

[7] R.C. Jammalamadaka, R. Gamboni, S. Mehrotra. "A middleware approach for building secure network drives over untrusted internet data storage". *Proc. ACM conference on Extending database technology: Advances in database technology*, 2008, pp. 710-714.

[8] E.J. Goh, H. Shacham, N. Modadugu and D. Boneh. "SiRiUS: Securing remote untrusted storage". *Proc. network and distributed system security*, 2003, pp. 131-145.

[9] M. Kallahalla, E. Riedel, and R. Swaminathan. "Scalable secure file sharing on untrusted storage". *Proc. USENIX conference on file and storage technologies*, 2003, pp. 29-42.

[10] A. Singh and L. Liu. "SHAROES: A Data Sharing Platform for Outsourced Enterprise Storage Environments". *Proc. ICDE*, 2008, pp. 993-1002.

[11] J. Yao, S. Chen, S. Nepal, D. Levy, and J. Zic. "TrustStore: Making Amazon S3 Trustworthy with Services Composition". *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGRID '10)*. 2010, pp. 600-605.

[12] J. Brodkin. "Gartner: Seven cloud-computing security risks". *Network World*, July 2, 2008.

[13] B. Panzer-Steindel. "Data Integrity. CERN/IT Draft 1.3 8". April 2007 http://indico.cern.ch/getFile.py/access?contribId=3&sessionId=0&resId=1&materialId=paper&confId=13797 (accessed Feb , 2011)

[14] G. Sivathanu, C.P. Wright, and E. Zadok. "Ensuring data integrity in storage: techniques and applications". *Proceedings of the 2005 ACM workshop on Storage security and survivability*. pp. 26-36.

[15] P.M. Chen, E.K. Lee, G.A. Gibson, R.H. Katz, and D.A. Patterson. "RAID: high-performance, reliable secondary storage". *ACM Comput. Surv.* vol 26, no 2, June 1994.

[16] D.A. Patterson, G. Gibson, and R.H. Katz. "A case for redundant arrays of inexpensive disks (RAID)". *SIGMOD Rec.* vol.17, no.3, June 1988, pp. 109-116.

[17] S. Patil, A. Kashyap, G. Sivathanu, and E. Zadok. "FS: An In-Kernel Integrity Checker and Intrusion Detection File System". *Proceedings of the 18th USENIX conference on System administration* (LISA '04). pp. 67-78.

[18] H. Krawczyk, M. Bellare, and R. Canetti. "HMAC: Keyed-hashing for message authentication". Technical Report RFC 2104, Internet Activities Board, February 1997.

[19] R. L. Rivest. "RFC 1321: The MD5 Message-Digest Algorithm". In *Internet Activities Board*. Internet Activities Board, April 1992.

[20] K.D. Bowers, A. Juels, and A. Oprea. " HAIL: a high-availability and integrity layer for cloud storage". In *Proceedings of the 16th ACM conference on Computer and communications security* (CCS '09). pp. 187-198.

[21] J. Whitehead. "A New RAID for Cloud-Scale Data Integrity . Computer Technology Review". *Intelligent Strategies and Solution for the Storage Industry*. Tuesday, 24 November 2009.

[22] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song. "Provable data possession at untrusted stores". *Proceedings of the 14th ACM conference on Computer and communications security* (CCS '07). pp. 598-609.

[23] G. Ateniese, R.D. Pietro, L.V. Mancini, and G. Tsudik. "Scalable and efficient provable data possession". *SecureComm '08*. 2008.

[24] Q. Wang, C. Wang, J. Li, K. Ren, and W. Lou. "Enabling public verifiability and data dynamics for storage security in cloud computing". *Proceedings of the 14th European conference on Research in computer security* (ESORICS'09), 2009, pp. 355-370.

[25] A. Juels and B. S. Kaliski, Jr.." Pors: proofs of retrievability for large files". *Proceedings of the 14th ACM conference on Computer and communications security* (CCS '07). 2007, pp. 584-597.

[26] C. Erway, A. Kupcu, C. Papamanthou, and R. Tamassia. "Dynamic provable data possession". *Proceedings of the 16th ACM conference on Computer and communications security* (CCS '09). 2009, pp. 213-222.

[27] C. Wang , Q. Wang , K. Ren , W. Lou. "Privacy-preserving public auditing for data storage security in cloud computing", *Proceedings of the 29th conference on Information communications*, p.525-533, March 14-19, 2010, San Diego, California, USA.

[28] L. Musthaler and B. Musthaler. "Enhanced trust and data integrity in the public cloud". *Network World*, December 03, 2010.

[29] J. Gray. "The transaction concept: virtues and limitations". *Proceedings of the seventh international conference on Very Large Data Bases* Volume 7 (VLDB '1981), Vol. 7. 1981, pp. 144-154..

[30] SNIA. "Cloud Data Management Interface (CDMI) specification". http://www.snia.org/tech_activities/publicreview/CDMI_Spec_v1.01f_DRAFT.pdf

[31] R. Perlman. "The Ephemerizer: Making Data Disappear", Sun *Microsystems Whitepaper*, 2005, http://research.sun.com/techrep/2005/smli_tr-2005-140.pdf (accessed on December 22, 2010)

[32] R. A Popa, and J. R. Lorch. "Enabling Security in Cloud Storage SLAs with CloudProof". *Technical report, Microsoft Research*. 2010

[33] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. "A view of cloud computing". *Commun. ACM* vol 53, no 4, April 2010, pp. 50-58.

[34] C. Cachin, I. Keidar, and A. Shraer. "Trusting the cloud". *SIGACT News* vol 40, no 2, June 2009, pp. 81-86.