Analysis of Integrity Vulnerabilities and a Non-repudiation Protocol for Cloud Data Storage Platforms

[†]Jun Feng*, [†]Yu Chen, [§]Wei-Shinn Ku, [‡]Pu Liu

[†]Dept. of Electrical & Computer Engineering, SUNY - Binghamton, Binghamton, NY 13902
[§]Dept. of Computer Science & Software Engineering, Auburn University, Auburn, AL 36849
[‡]Systems and Technology Group, IBM Endicott, Endicott, NY 13760

***Abstract* - Data storage technologies have been recognized as one of the major dimensions of information management along with the network infrastructure and applications. The prosperity of cloud computing requires the migration from server-attached storage to network-based distributed storage. Along with variant advantages, distributed storage also poses new challenges in creating a secure and reliable data storage and access facility. The data security in cloud is one of the challenges to be addressed before the novel pay-as-you-go business model can be accepted and applied widely. Concerns are raised from both insecure/unreliable service providers and potential malicious users. In this article, we analyze the integrity vulnerability existing in the current cloud storage platforms and show the problem of repudiation. A novel non-repudiation (NR) protocol specifically designed in the context of cloud computing environment is proposed. We have also discussed the robustness of the NR protocol against typical attacks in the network environments.**

***Keywords:* Cloud Computing, Data Integrity, Storage Security, Non-Repudiation.**

1. Introduction

Leading to a revolutionary change in the computing services, cloud computing has been viewed as the future of the IT industry. It has gained great attention from both industry and academia since 2007. As a further step on top of Grid Computing, cloud computing aims to provide users more flexible services in a transparent manner – all services are allocated in a "cloud" that actually is a collection of devices and resources connected through the Internet. However, there are challenges to be addressed before the beautiful blueprint is accepted widely. One of the most impending tasks is the security of data storage in the cloud.

As shown in Figure 1, cloud computing can fundamentally provide services via the Internet, through which to access large amounts of data and computing resources. Although the definition of cloud computing is not clear yet, several pioneer commercial platforms have been constructed and are open to the public, such as Amazon's Computer Cloud AWS (Amazon Web Service) [1] and Microsoft Azure Service Platform [5].

*Manuscript submitted on June 12, 2010 to *the 2nd International Workshop on Security in Cloud Computing (SCC 2010), held in conjunction with the 39th International Conference on Parallel Processing* (ICPP 2010), San Diego, California, USA, Sept. 13, 2010. Corresponding author: Jun Feng, SUNY – Binghamton, Binghamton, NY 13902. E-mail: jfeng3@binghamton.edu, Tel.: (607) 239-3165.

The IDC survey in August 2008 has shown that security is the most serious concern that customers have ascribed to the cloud computing [3]. Meanwhile, the research in cloud computing security is far from mature [4].

First of all, the uniqueness of the security issues in cloud computing is not recognized. Some researchers still believe that cloud computing security is not much different from existing security practices and the security aspects can be well managed using the traditional techniques such as digital signature, encryption, firewall, and/or the isolation of virtual environments, etc. [4]. For example, SSL (Secure Sockets Layer) is a protocol that provides reliable secure communications on the Internet for applications as web browsing, e-mail, instant messaging and other data transfers in the current Internet.

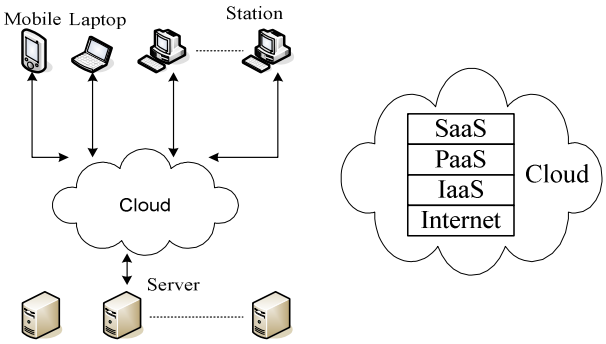


Figure 1. Illustration of cloud computing principle.

Secondly, the specific security requirements in cloud computing are still cloudy to the community. Nevertheless, cloud security is an ongoing important area of research. Many consultants and security agencies have issued warnings on security threats in the cloud computing model [2]. Besides, cloud consumers still wonder whether the cloud is secure. There are at least two concerns when they use the cloud. One is the users who do not want to reveal their data to cloud service providers. It is because of the sensitivity of the data (e.g. patient records). Another is the integrity of the data that consumers receive from the cloud. In fact, for cloud security, it requires more than conventional security mechanisms.

This paper analyzes the data integrity vulnerabilities in today's commercial cloud storage platforms in section 2, such as Amazon's AWS [1], Microsoft Windows Azure [5], and Google App Engine (GAE) [9]. After discussing

couples of simple mechanisms that can make up the missing link between the uploading and downloading procedure in section 3, we propose a novel non-repudiation (NR) protocol to address the problem in section 4. In section 5 we discuss the robustness of our NR protocol against several typical attacks in network environments. Section 6 concludes this paper with a discussion of our future work.

2. Study of Existing Vulnerabilities

In normal network based applications, the user authentication, data confidentiality and integrity issues can be solved through IPSec proxy using encryption and digital signature, and the key exchange can be solved by SSL proxy. Such methods have been applied to today's cloud computing to secure data and communication and the service providers claim that their services are secure. This section describes the secure methods used in the cloud services and points out their vulnerabilities on data integrity.

2.1 Storage Service on the AWS

Amazon provides *Infrastructure as a Service* (IaaS) with different names, such as *Elastic Compute Cloud* (EC2), *SimpleDB*, *Simple Storage Service* (S3), and so on. They are supposed to ensure the confidentiality, integrity, and availability of customers' applications and data. Figure 2 presents one of the data processing approaches adopted in Amazon's AWS [1]. This method is used to transfer large amounts of data between the AWS storage cloud and portable user devices.

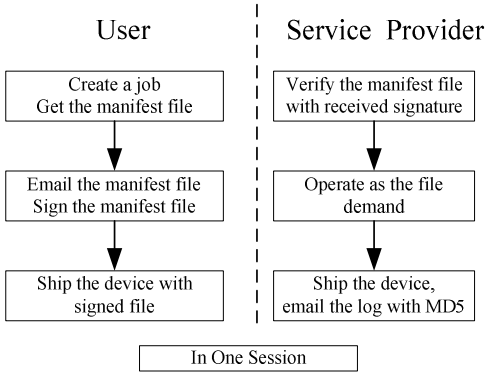


Figure 2. An AWS data processing approach.

When the user wants to upload the data, he/she needs to store some parameters such as AccessKeyID, DeviceID, Destination, etc. into an import metadata file called *manifest file*, then signs the manifest file, and emails the signed manifest file to Amazon. Another metadata file named *signature file* is used by the AWS to describe the cipher algorithm that is adopted to encrypt the job ID and the bytes in the manifest file. Using the signature file the service provider can uniquely identify and authenticate the user request. The signature file is attached with the storage device, which is shipped to Amazon for efficiency.

On receiving the storage device and the signature file, the service provider will validate the signature in the device with the manifest file obtained through the email. Then, Amazon will email management information back to the user including the number of bytes saved, the MD5 of the bytes, the status of the load, and the location on Amazon S3 of the AWS Import Export Log. The log contains details about the data files that have been uploaded, including the key names, number of bytes, and MD5 checksum values.

The downloading process is similar to the uploading process. The user creates manifest and signature file, emails the manifest file and ships the storage device attached with signature file. When Amazon receives them, it will validate the two files, copy the data into the storage device, ship it back, and email the user the status including MD5 of the Data. Amazon has claimed that the maximum security is obtained via SSL endpoints [1].

2.2 Storage Service on the Azure

The Windows Azure Platform (Azure) is an Internet-scale cloud services platform hosted at Microsoft data centers. The centers provide an operating system and a set of developer services that can be used individually or collectively [5]. Azure Platform provides scalable storage service. There are three basic data items: Blobs (up to 50GB), Tables, and Queues (<8k). In the Azure storage, based on the Blob, Table, and Queue operations, Microsoft promises to achieve confidentiality of users' data. The procedure shown in Figure 3 provides security for data accessing to ensure that the data will not be lost.

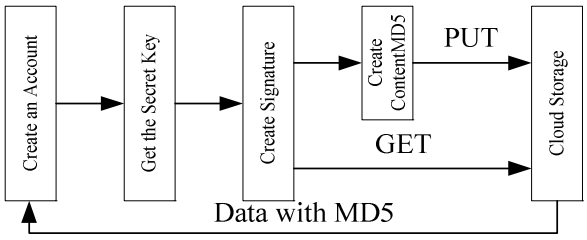


Figure 3. Security data access procedure.

When using Windows Azure Storage service, a user needs to create an account, which can be obtained from the Azure portal web interface. After creating an account, the user will receive a 256-bit secret key. Each time when the user uploads the data to or downloads the data from the cloud, he/she has to use the secret key to create a HMAC SHA256 signature for each individual request for identification. Then the user uses signature to authenticate request at server. The signature is passed with each request to authenticate the user requests by verifying the HMAC signature.

The example in Table 1 is a REST request for a PUT/GET block operation of a data block [7]. Content-MD5 can be provided to guard against network transfer errors and data integrity. The Content-MD5 in the PUT is the MD5 checksum of the data block in the request. The MD5 checksum is checked by the server. If it does not match, an error is returned. The content length specifies the size of the

data block contents. There is also an authorization header inside the HTTP request header as shown in Table 1.

At the same time, if the Content-MD5 request header was set when the Blob has been uploaded, it will be returned in the response header. Therefore, the user can check for message content integrity. Additionally, the secure HTTP connection is used for true data integrity [8].

Table 1. Example of a REST request.

PUT http://jerry.blob.core.windows.net/movie/mov.avi ?comp=block &blockid=BlockId1 &timeout=30 HTTP/1.1 Content-Length: 2174344 Content-MD5: FJXZLUNMuI/KZ5KDcJPcOA== Authorization:SharedKeyjerry:F5a+dUDvef+PfMb4T8Rc2jHcwfK58KecSZY+l2naIao= x-ms-date: Sun, 13 Sept 2009 22:30:25 GMT x-ms-version: 2009-04-14
GET http://jerry.blob.core.windows.net/movies/mov.avi HTTP/1.1 Authorization:SharedKeyjerry:ZF3lJMtkOMi4y/nedSk5Vn74IU6/fRMwiPsL+uYSDjY= x-ms-date: Sun, 13 Sept 2009 22:40:34 GMT x-ms-version: 2009-04-14

2.3 SDC Service on the GAE

The Google App Engine (GAE) [9] provides a powerful distributed data storage service that features a query engine and transactions. An independent third party auditor, who claimed that GAE can be secure under the SAS70 auditing industry standard, issued Google Apps a qualified SAS70 Type II certification. However, from its on-line storage technical document of lower API [12], there are only some functions such as GET and PUT. There is no content addressing the issues of securing storage services. The security of data storage is assumed guaranteed using techniques such as SSL link, based on our knowledge of security methods adopted by other services.

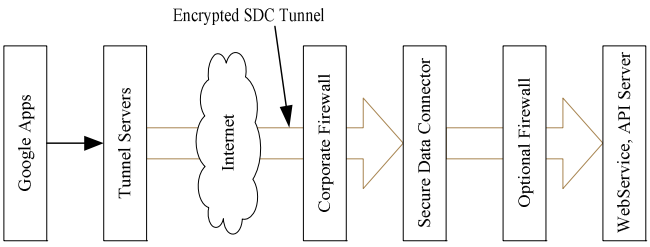


Figure 4. Illustration of Google SDC Working flow.

Figure 4 is one of the secure services, called Google Secure Data Connector (SDC), based on GAE [10]. The SDC constructs an encrypted connection between the data source and the Google Apps. As long as the data source is in the Google Apps domain, when the user wants to get the data, he/she will firstly send an authorized data requests to Apps, which forwards the request to the Tunnel Server. The tunnel servers validate the request. If the identity is legal, the tunnel protocol allows the SDC to set up connection, authenticate, and encrypt the data that flows across the Internet. At the same time, the SDC uses resource rules to verify whether a user is authorized to access a specified resource. When the request is valid, SDC performs a network request. The service sever validates the signed request, checks the credentials, and returns the data if the user is authorized.

The SDC and Tunnel Server function as the proxy to encrypt connectivity between Apps and the internal network. In addition, for higher security requirements, the SDC adopts signed request to add authentication information to requests that are made through SDC. The signed request has to include the owner_id, viewer_id, instance_id, app_id, public_key, consumer_key, nonce, token, signature to the request [11] to ensure the integrity, security and privacy.

2.4 Integrity Vulnerability

Previous subsections have introduced the security design of three different commercial cloud data storage platforms. For large storage (> 1TB) service, it requires a convenient shipping method to deliver the storage device to the service provider; for smaller size (≤ 50 GB), the data can be uploaded or downloaded via the Internet connections. To provide date integrity, the Windows Azure Storage Service stores the uploaded data MD5 in the database and returns it to the user when the user wants to retrieve the data. In contrast, the Amazon AWS computes the data MD5 and emails to the user for the integrity checking. The SDC is based on GAE to strengthen the Internet authentication by a signed request. Figure 5 shows the common approach of the three platforms although their behaviors are different.

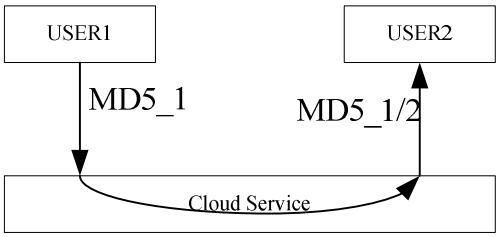


Figure 5. Illustration of potential integrity problem.

As shown in Figure 5, when user1 stores data in the cloud, he/she sends the data to service providers with a MD5, which is labeled as MD5_1 in Fig. 5. If the data is transferred through the Internet, a signed request is included to enforce the authentication. When the service provider receives the data and the MD5_1, it stores the data with the MD5_1. When the service provider gets a verified request to retrieve the data from another user or the original user, it will send the data with a MD5 to the user. On the Azure platform, the original MD5_1will be sent, in contrast, a re-computed MD5_2 is sent on Amazon's AWS.

The procedure is secure regarding each individual session. The integrity of the data in the transmission can be guaranteed by the SSL protocol. However, from the perspective of the cloud storage service, the data integrity depends on not only the security of both the uploading and downloading sessions, but also the protection of the data in the storage medium. The uploading phase can only ensure the data received by the cloud storage is the data user uploaded; the downloading phase can guarantee the data user retrieved is the data cloud storage sent. Unfortunately, this procedure cannot guarantee the data is not modified in the storage space.

Let's consider the following scenario. Assume that Alice, a company CFO, stores the company financial data at a cloud storage service provided by Eve. And then Bob, the company administration chairman, downloads the data from the cloud. There are three important concerns raised in this simple procedure:

1. Confidentiality: Although she is the storage service provider and has full access to the data, Eve is considered as an untrustworthy third party and Alice and Bob do not want reveal the data to her.
2. Integrity: As the administrator of the storage service, Eve has the capability to play with the data in hand. How can Bob be confident that the data he fetched from Eve is the same as what sent by Alice? Are there any measures to guarantee that the data has not been tampered by Eve?
3. Repudiation: If Bob finds the data has been tampered, is there any evidence to demonstrate that it is Eve who should be responsible for the fault? Similarly, Eve also needs certain evidence to prove her innocence.
4. Blackmail: The repudiation problem also opens a door for blackmailers. Alice stores some data in the cloud, and later she downloads it. Then, she reports that her data were tampered due to the fault of the storage provider. Alice claims compensation for her so-called loss. Now, the service provider needs evidence to demonstrate her innocence.

Recently, a potential customer asked a question on a cloud mailing-group regarding data integrity and service reliability. The reply from the developer was "*We won't lose your data - we have a robust backup and recovery strategy - but we're not responsible for you losing your own data …*" [6]. Obviously it is not persuasive enough to make the potential customer be confident with the service.

In addition, the repudiation issue opens a door for blackmailers when the user turned to be malicious. Assume that Alice wants to blackmail Eve, a cloud storage service provider who claims that the data safe and integrity as their key features. Alice stores some data in the cloud, and later she downloads the data. Then, she reports that her data were broken and that is the fault of the storage provider. Alice claims compensation for her so-called loss. How can the service provider demonstrate her innocence?

The confidentiality can be achieved by adopting robust encryption schemes. However, the integrity and repudiation are not handled well on the current cloud storage platform. One-way SSL session only authenticates the one-way integrity. One critical link is missing between the uploading and downloading sessions: there is no mechanism for the user/service provider to check whether the record is modified in the cloud storage. This vulnerability leads to two following questions that need to be answered:

- **Upload-to-Download Integrity:** since the integrity in uploading and downloading phase are handled separately, how can the user/provider know the data retrieved from the cloud is the same data that the user uploaded previously?
- **Repudiation between users and service providers:** When data error happened without the transmission error in the uploading/downloading phase, how can the user/service provider prove his/her innocence?

3. Bridging the Missing Links

This section presents several preliminary ideas based on digital signature and authentication coding schemes. Based on whether there is a third authorities certified (TAC) by the user and provider, and whether the secret key sharing technique (SKS) is adopted there are four solutions to bridge the missing link of the data integrity between the uploading and downloading procedures. Actually, other digital signature technologies can be adopted under this framework to fix this vulnerability with different approaches.

3.1 Neither TAC nor SKS

	Uploading Session
1:	User: sends data to service provider with MD5 and MD5 Signature by User (MSU);
2:	Service Provider: verifies the data with MD5, if it is valid, service provider sends back the MD5 and MD5 Signature by Provider (MSP) to user;
3:	MSU is stored at the user side, and MSP is stored at the service provider side.

Once the uploading operation finished, both sides agreed on the integrity of the uploaded data, and each side owned the MD5 and MD5 signature generated by the opposite site.

	Downloading Session
1:	User: sends request to service provider with authentication code;
2:	Service Provider: verifies the request identity, if it is valid, service provider sends back the data with MD5 and MD5 Signature by Provider (MSP) to user;
3:	User verifies the data using the MD5.

When disputation happens, the user or the service provider can check the MD5 and the signature of MD5 generated by opposite side to prove its innocence. However, there are some special cases exist. When the service provider is trustworthy, only MSU is needed; when user is

trustworthy, only MSP is needed; if each of them trusts the other side, neither MSU nor MSP is needed. Actually, that is the method adopted in the current cloud computing platforms. Essentially, this approach implies that when the identity is authenticated, the trust is established.

3.2 With SKS but without TAC

	Uploading Session
1:	User: sends data to service provider with MD5;
2:	Service Provider: verifies the data with MD5, if it is valid, the service provider sends back the MD5;
3:	The service provider and the user share the MD5 with SKS.

Then, both sides agree on the integrity of the uploaded data, and they share the agreed MD5, which is used when disputation happens.

	Downloading Session
1:	User: sends request to the service provider with authentication code;
2:	Service Provider: verifies the request identity, if it is valid, the service provider sends back the data with MD5;
3:	User verifies the data through the MD5.

When disputation happens, the user or the service provider can take the shared MD5 together; recover it and prove his/her innocence.

3.3 With TAC but without SKS

	Uploading Session
1:	User: sends data to the service provider along with MD5 and MD5 Signature by User (MSU);
2:	Service Provider: verifies the data with MD5, if it is valid, the service provider sends back the MD5 and MD5 Signature by Provider (MSP) to the user;
3:	MSU and MSP are sent to TAC.

On finishing the uploading phase, both sides agree on the integrity of the uploaded data, and TAC owns their agreed MD5 signature.

	Downloading Session
1:	User: sends request to the service provider with authentication code;
2:	Service Provider: verifies the request with identity, if it is valid, the service provider sends back the data with MD5;
3:	User verifies the data through the MD5.

When disputation happens, the user or the service provider can prove its innocence by presenting the MSU and MSP stored at the TAC.

Similarly, there are some special cases. When the service provider is trustworthy, only the MSU is needed; when the user is trustworthy, only the MSP is needed; if each of them trusts the other, the TAC is not needed. Again, the last case is the method adopted in the current cloud computing platforms. When the identity is authenticated, the trust is established.

3.4 With both TAC and SKS

	Uploading Session
1:	User: sends data to the service provider with MD5;
2:	Service Provider: verifies the data with MD5;
3:	Both the user and the service provider send MD5 to TAC;
4:	TAC verifies the two MD5 values. If they match, the TAC distributes MD5 to the user and the service provider by SKS.

Both sides agree on the integrity of the uploaded data and share the same MD5 by SKS, and TAC own their agreed MD5 signature in demand.

	Downloading Session 1
1:	User: sends request to the service provider with authentication code;
2:	Service Provider: verifies the request identity, if it is valid, the service provider sends back the data with MD5;
3:	User verifies the data through the MD5.

When disputation happens, the user or the service provider can prove their innocence by checking the shared MD5 together. If the disputation cannot be resolved, they can seek further help from the TAC for the MD5.

There are special cases. When the service provider is trustworthy, only the user needs the MD5; when the user is trustworthy, only the service provider needs MD5; if both of them can be trusted, the TAC is not needed. That is the method used in the current cloud computing platform.

4. A Novel Non-repudiation Protocol

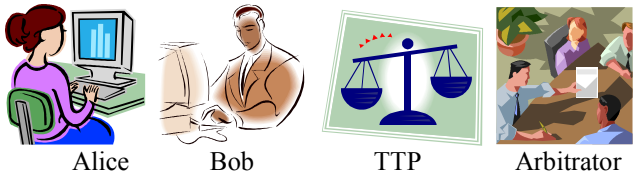
There are many reported efforts on *Non-Repudiation* (NR) under the standard framework in the past [13]. However, although some ideas are helpful to solve the problem, the traditional NR protocols consist of at least four steps and they are designed for the traditional Internet, and not optimal for the cloud services.

4.1 Overview to Non-Repudiation Protocol

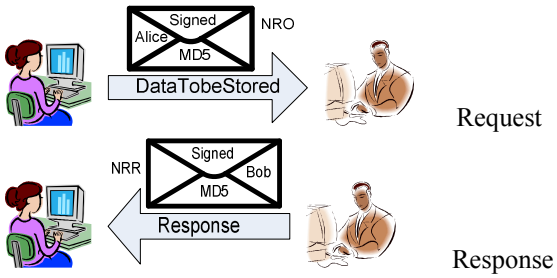
This section describes a new non-repudiation protocol for cloud storage. The idea of integrity check and non-repudiation is not new, which can be found in literatures. To date, however, there is no reported effort that integrates them to solve the problem in cloud storage platforms.

To eliminate the reputation between Alice and Bob, we propose to bridge the two sessions with an integrity link based on a new fair non-repudiation protocol. The solution covers also the uploading session and downloading session. In addition to the normal process, we also provide sub-protocols that allow Alice to complete or abort the execution of protocol, without waiting for responses from the other potentially malicious party.

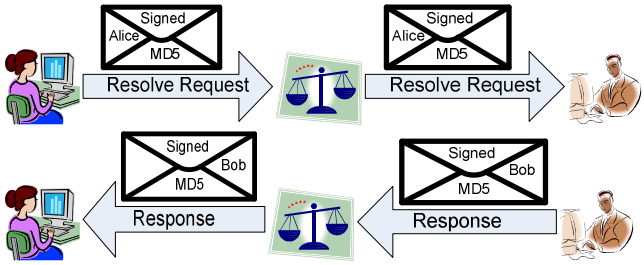
Figure 6(a) shows that there are four roles in the NR protocol: Client (Alice), Cloud Storage Provider (Bob), Trusted Third Party (TTP), and an Arbitrator. Assuming, in most of the cases, Alice and Bob are honest and willing to complete the transactions by themselves and the TTP is only initiated as the last course. None of the parties is going to act against its own interests.



(a) Roles in the protocol



(b) Off-line TTP



(c) In-line TTP

(d) Disputation Arbitrate

Figure 6. Illustration of the Two-Party Non-Repudiation (TPNR) Protocol Work Flows

In each transmission, the sender (Alice/Bob) must attach certain extra information, with the message, collectively called *evidence* since its function is to settle repudiation, when appears. For Alice, it is called "non-repudiation of origin" (NRO); for Bob, it is named "non-repudiation of Receipt" (NRR).

For confidentiality, the sender encrypts the evidence with the recipient's public key. To prevent replay attacks, a random number and a sequence number are included. The sequence number increases one by one. In case that someone refuses to accept the message, we add a time limit and a sub-protocol to resolve this. Besides a flag to label the process, we also include the IDs of the following in the plaintext message for convenience: the sender, the recipient, and the TTP. The evidence consists of the hash results of these IDs and the hash of the data.

To achieve non-repudiation, we require the sender sign the hash value with her/his private key. Then the evidence is *Encrypt{Sign(HashofData), Sign(Plaintext)}*. After one transaction is completed, Alice will get *Non-Repudiation of Receipt* (NRR) from Bob; Bob will get *Non-Repudiation of Origin* (NRO) from Alice. The peers should check the consistency between the hash of the plaintext and the plaintext at first.

The NR protocol takes advantages of the signed integrity checking of the data in the evidence. Not only facilitate detecting data tampering, the signature of the sender also makes it impossible for the sender to deny his/her activity. Meanwhile, being encrypted with the public key of the recipient, the evidence guarantees the consistence of the hash with the plaintext.

For instance, Alice owns the NRR signed by Bob, and she can send it to him when she downloads the data. This is helpful to avoid repudiation when the downloaded data has been tampered. When there is an inconsistency, they have to submit the signed integrity to the arbitrator. When Alice and Bob are honest and the network functions well, they can exchange message in Normal Mode which work on off-line TTP as shown in Figure 6(b).

The requirement of evidence can guarantee the non-repudiation. However, in practice not all users or service providers are willing to completely obey the rules set by the protocol. The honest party will suffer the unfairness if there is no mechanism to protect them. In order to ensure fairness, it is expected that once a user/service provider has sent his/her evidence to the peer, it is guaranteed that he/she will receive the evidence from the peer. Otherwise, none of them can hold the evidence from the other side without sending out her/his own. For instance, if Bob, the service provider, does not respond after he has received the NRO from Alice, then Alice will be in a disadvantage position in this business with Bob.

To provide a fair working environment, two more models are implemented in the NR protocol: Abort model and Resolve model.

4.2 Abort model

The Abort model makes it possible that Alice is able to cancel the transaction under certain scenarios. As shown in Figure 6(b) the Abort model works in the off-line TTP mode. To terminate an ongoing data transferring procedure and quit from an undesired situation, Alice is only required to send Bob the transaction ID with the NRO. A TTP is not necessary to finish the abort process. This is different from the traditional non-repudiation protocol. Only when the message cannot be delivered, the TTP is needed.

On receiving an Abort request from Alice, Bob should verify its consistency first. If the request is valid, Bob will respond with "Accept" or "Reject" along with a NRR. Otherwise, Bob will send an "Error" message that request

Alice double check the parameters included in the Abort request, regenerate it, and re-submit the request.

4.3 Resolve model

More complicated scenarios, in which the data transaction agreement cannot be achieved without the interference from the trusted third party, are possible in either the normal data backup procedure or in an Abort operation. For instance, in the middle of a data transaction, Alice did not receive the response from Bob within the pre-set time-out limit. There are multiple reasons that may lead to such an anomaly, e.g., the link between Alice and Bob is broken, her request was dropped and Bob has never received, or Bob is malicious and he did not respond with her requests. It is beyond Alice's capability to solve the problem and requires a reliable third party that plays the role of arbiter.

Figure 6(c) illustrates the functionality of the Resolve mode, which works as an in-line TTP. In this scheme, the TTP could be a reliable server, or a reliable service provider other than Bob. Alice sends the transaction ID, the NRO, and a report of anomalies to TTP, and requires the TTP's help to resume the disrupted data transferring process. Once the TTP verifies the genuineness and the consistency, the TTP will generate the Resolve request to the recipient along with a time stamp.

Assume the communication channels among TTP, Alice, and Bob are reliable. Alice sends Bob a resolve message to initiate the resolve procedure. The resolve message carries the query along with the information such as transaction ID, Alice's user ID, etc. After checking the contents of the query and verifying the consistency, Bob will respond to Alice through TTP with the NRR along with the actions he will take according to the status. For example, Bob may agree to continue the transaction; or, he may require Alice to restart the session. If Bob does not reply the Resolve query from TTP before time out, the TTP will respond to Alice by telling her that this session is failed and Bob did not respond.

The Resolve model functions as the kernel of our Non-Repudiation protocol. When Alice has to stop the transaction, she needs evidence to protect herself in case of repudiation. For this purpose, the NRR from Bob and information from the TTP are critical. At the same time, Bob cannot get any benefits when he maliciously refuses to reply. In this design, we do not consider the data exchanging through TTP in the context of Cloud storage services. Normally the size of the data set is very large, which is not feasible to be stored and/or forwarded by the TTP.

Meanwhile, if Alice has sent the NRO and has not received the NRR before the time out, she can initiate the Resolve mode. Here, Alice hopes that TTP can help to make the transaction continue, and she wants to get the NRR from Bob through triggering the resolve procedure.

If Bob sends NRR to Alice, and he has not received the response from Alice directly before time out or has not received Resolve from TTP. Bob may suppose that Alice has agreed on the NRR, or Bob can initial a resolve procedure at the TTP.

4.4 Summary

There are three models in the Non-Repudiation protocol: Normal model, Abort model, and Resolve model. Normal and Abort models work in off-line TTP mode as shown in Figure 6(b). Resolve model works in in-line TTP mode as shown in Figure 6(c). If disputation happens, the Arbitrator can ask Alice and Bob to provide evidence for judging as shown in Figure 6(d).

Compared with the traditional non-repudiation protocol, the main advantage of our protocol lies in the efficient design for the Cloud platforms. For instance, in the Normal and Abort models, it takes Alice and Bob merely two steps without TTP to exchange messages and non-repudiation evidence directly. In contrast, the same operation takes four steps in the traditional non-repudiation protocol.

When Alice cannot get the non-repudiation evidence from Bob, TTP will be invoked in the Resolve model. From the perspective of Bob, the situation is simpler. If he did not receive the NRO, he is not required to take any action since most of the transactions are initiated by Alice. Only when there is no further response or specified following activities after he has sent NRR, Bob needs to initiate the Resolve procedure in case disputation happens.

5. Robustness of the NR Protocol

This section provides a brief analysis of the security of our NR protocol under five well known attacks [14].

5.1 Man-in-the-middle attack

The classic Man-in-the-middle attack (MITM) is a form of active eavesdropping in which the attacker makes independent connections with the victims and relays messages between them. The attacker can intercept all messages going between the two victims and inject new ones.

However, MITM attack can succeed only when the attacker can impersonate the end parties. It can be prevented by the authentication. In this protocol, when the party gets the other's public key, they should authenticate the validity to avoid the MITM.

5.2 Reflection attack

A reflection attack is a method of attacking a challenge-response authentication system that uses the same protocol in both directions.

Our protocol is not a challenge-response authentication system; furthermore, each message contains a unique identifier. Consequently, the reflection attack is avoided.

5.3 Interleaving attack

The interleaving attack is similar to man-in-the-middle attack, but it can attack the protocol in which all parties have authentic copies of all others' public keys.

Interleaving attack can possibly succeed when there are several rounds to exchange key and the to-and-from messages are symmetrical or the symmetric key establishment is on the shared session key. In this protocol, the message is not symmetrical and binding with a unique sequence number. In addition, each session is finished only in one round. Therefore, the interleaving attack is avoided.

5.4 Replay attack

A replay attack is a form of network attack in which a valid data transmission is maliciously or fraudulently repeated or delayed. This is carried out either by the originator or by an adversary who intercepts the data and retransmits it.

Replay attack can be avoided by the use of challenge-response techniques and embedding target party ID in response or the timestamp. In this protocol, we use unique sequence number with the sender signature to avoid the attack. If someone intercepts the message and replays it to TTP, even the attacker can modify the sequence number in the plaintext, the attacker cannot modify the Encrypted Hash value protected by the sender's private key.

5.5 Timeliness attack

Without deadline, the protocol does not know when the step is terminated, which can introduce some problems. For fairness, each party can stop the execution after a finite time. In this protocol, we add a time limit field into the message in order to limit the reception time of a message [15].

6. Summaries and Future Work

Cloud Computing has been considered as the future of IT that leads to novel computing models. However, security has been one of the major concerns that prevent commercial applications from being accepted widely.

This paper reports our current work on data security in cloud computing environments. We have revealed the existing vulnerabilities in today's commercial cloud service platforms due to the missing of connection between the uploading and downloading phases. We briefly introduced four solutions based on the digital signature and the authentication code. At present time we cannot tell which is the most suitable to be implemented in practice, more experimental study will be conducted.

A two-party non-repudiation protocol bas been proposed according to the cloud conditions. The TPNR protocol is like the TCP/IP three-phase handshaking protocol. It is designed to exchange the evidence in the data transaction, which removes the ambiguities that lead to repudiations or disputations between the user and service provider.

The idea of integrity check and non-repudiation is not new, which have been studied thoroughly in different contexts. However, it is a new direction to integrate both of them to protect data in cloud storage services. Although the traditional non-repudiation protocol can be applied in the cloud storage environments, it takes more steps to complete the whole transaction compared to our protocol. To date, we have not seen any reported research on how to bridge the integrity link based on the non-repudiation protocol.

Cloud storage is only attractive to large volume (TB) data backup. The current cloud storage services, such as S3, normally adopt the surface mail as the ship method (FedEx, etc). Therefore, the time required for executing the protocol is really trivial comparing to the time consumed by delivering the storage devices by surface mail. Additionally, there are many other issues that affect the performance, such as the storage medium type, system architecture, security algorithm, etc. Thus, we leave the experimental study of performance evaluation as our next step work.

References

[1] Amazon Import/Export Developer Guide Version 1.2, http://aws.amazon.com/documentation/, August 2009.

[2] J. Heiser and M. Nicolett, "Assessing the Security Risks of Cloud Computing," *Gartner* Inc., June 2, 2008.

[3] F. Gens, "IDC on 'the Cloud': Get Ready for Expanded Research", http://blogs.idc.com/ie/?p=189, Sept. 23, 2008.

[4] K. M. Khan, "Security Dynamics of Cloud Computing," *CUTTER IT JOURNAL*, June/July 2009, p38-43

[5] Microsoft Azure Services Platform, http://www.microsoft.com/azure/default.mspx, 2009.

[6] Google mail, http://groups.google.com/group/google-appengine/browse_thread/thread/782aea7f85ecbf98/8a9a505e8aaee07a?show_docid=8a9a505e8aaee07a#

[7] azura word, http://msdn.microsoft.com/en-us/azure/cc994380.aspx, 2009.

[8] azura MSDN API, http://msdn.microsoft.com/en-us/library/dd179394.aspx, 2009.

[9] What is Google App Engine? http://code.google.com/appengine/docs/whatisgoogleappengine.html, 2009.

[10] GoogleSecure Data Connector Developer's Guide: Overview, http://code.google.com/securedataconnector/docs/overview.html

[11] Introduction to Signed Request, http://wiki.opensocial.org/index.php?title=Introduction_To_Signed_Requests

[12] Package Summary, com.google.appengine.api.datastore, http://code.google.com/appengine/docs/java/javadoc/com/google/appengine/api/datastore/package-summary.html, 2009.

[13] J. Onieva, J. Lopez, and J. Zhou. "Secure Multi-Party Non-repudiation Protocols and Applications". ISBN 978-0-387-75629-5, *Advances in Information Security Series*, Springer, 2009.

[14] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, "Handbook of Applied Cryptography," 1996, CRC Press.

[15] P. Louridas, "Some Guidelines for Non-repudiation Protocols," Computer Communication Review, vol 30, no 5, October 2000.