A Fair Multi-Party Non-Repudiation Scheme for Storage Clouds

Jun Feng, Yu Chen, Douglas H. Summerville
Dept. of Electrical & Computer Engineering, Binghamton University (SUNY), Binghamton, NY 13902
{jfeng3, ychen, dsummer}@binghamton.edu

ABSTRACT

Data storage is one of the most profitable applications in Clouds. Although a transparent service model is convenient, it may be subject to the loss of data integrity. Our study revealed vulnerabilities in some commercial Cloud storage services. We analyzed the repudiation problem in a Cloud environment. In this paper, we propose a new multi-party non-repudiation (MPNR) scheme to fix the issue. Rationale behind the new scheme and a description of its operation are provided. We also discussed its robustness against typical network attacks.

KEYWORDS: Non-repudiation, Cloud Storage.

1. INTRODUCTION

Storage of data is potentially the most profitable application in the Cloud. Given this large potential profit for such services, cloud storage service providers attempt to persuade their users to store important and sensitive data to Cloud. They often advertise the convenience that this new business model offers to users.

A number of consultants and security agencies, however, have issued warnings concerning security threats in Cloud models [12] Potential users are left to wonder whether the confidentiality, integrity, and the availability of their data are guaranteed in Cloud Storage. Just as no one would want to put his valued possessions in a house without a secure lock, users are reluctant to move important and sensitive data to Cloud until these challenges have been well addressed. Therefore, to date this potentially valuable service model is still not widely accepted.

In addition, some user concerns cannot be alleviated by simply developing new technologies. There is naturally some psychological anxiety when a user is faced with the decision to store sensitive data in a location that is out of his control. In fact, some problems require more than conventional cyber security mechanisms and introduce new security challenges. Research in Cloud storage security is far from mature, and traditional cyber security solutions cannot provide enough protection in the Cloud for other reasons [25] .

Meanwhile, the uniqueness of secure Cloud storage still has not been fully understood. For example, one characteristic of Cloud Storage is mass storage in which data is communicated not only through the Internet, but also can be shipped by couriers or by other means if the size of data is huge, e.g. 1TB. Time is not a deterministic factor.

In this paper, we investigate current commercial Cloud storage platforms [29] [30] and some proposed architectures and approaches for secure Cloud storage. As discussed in detail in Section 2, there still exist vulnerabilities that can potentially lead to disputation. To correct such weakness, we propose a new fair multi-party non-repudiation (MPNR) scheme for secure Cloud storage systems. We focus on how to ensure integrity with fair non-repudiation, not just how to maintain integrity itself since current integrity algorithms are sufficient. The idea of integrity checks and non-repudiation is not new. Using traditional non-repudiation protocols, the receivers can decrypt the received data. This is not preferable for cloud storage. In cloud storage, the cloud provider is a hollow man-in-the-middle, it cannot access the content of data. Another issue is privacy, which is beyond the scope of this paper and interested readers are referred to reference [33].

The rest of this paper is structured as follows: Section 2 is related work. Section 3 presents the design of a new MPNR framework. Section 4 discusses the robustness of our scheme against typical malicious attacks. Finally, Section 5 summaries this paper.

2. RELATED WORK

We analyze vulnerabilities in today's commercial Cloud storage systems first, which open the door for potential data integrity tampering and disputation problems. Then, an introduction to several secure architectures for Cloud storage platforms and security approaches adopted in these architectures are described. Finally we explain why a fair MPNR is needed in this context.

Suppose there are three entities involved in Cloud storage, namely the data owner, the service provider and the users. The owner stores important or sensitive data to the Cloud and pays for this service. The service provider provides secure storage services and obtains profit. The users fetch data from Cloud storage and also pay for the service. Only the owner can decide and change the access control polices for his data. We also suppose that no party is trustworthy in the Cloud environment. That is, the owner, provider or users may deny their actions if doing so would compromise their interests, or they are willing to attack others if such behavior is beneficial to them. To date, no research has been done on fair non-repudiation for Cloud storage. Related work can be divided into three parts.

Cloud storage is an application that covers a number of services (SaaS, PaaS, IaaS, etc) [22]. The lifecycle of data in Cloud storage can be divided into three phases: Uploading, Maintenance, and Downloading. While data integrity in the uploading and downloading phases is achieved by cryptographic protocols such as SSL (Secure Sockets Layer) or TLS (Transport Layer Security), it is more complicated for users to monitor integrity and availability of remotely.

2.1 Existing Platforms for Cloud Storage

There currently exist some Cloud storage platforms (Amazon S3, Microsoft Azure, etc.). For large blocks of data (> 1TB), service providers such as Amazon AWS [29] require data to be shipped on a storage medium (e.g. a hard drive) while some extra authentication/authorization information is delivered through email. For smaller blocks of data (≤ 50 GB), upload or download via the Internet is used, just as Microsoft Azure [29] does.

Although the software may be different, a similar strategy has been used to provide data integrity (e.g. MD5 digest). As shown in Figure 1, when the owner uploads data into Cloud storage space, it can ship or send data to service providers with a digital digest, MD5_1. If the data is transferred through the Internet, a signed non-repudiation request could be used to ensure that data has not been tampered.

When the service provider receives the data with a signed

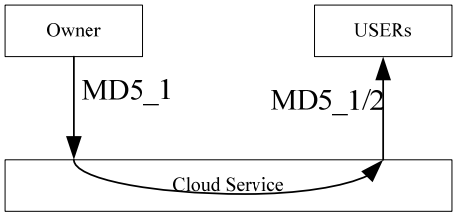


Figure 1. Illustration of potential integrity problem.

MD5, it stores data with the corresponding MD5_1. When the service provider gets a verified request to retrieve data from the users, it will send/ship the data with an MD5 to the user. On Amazon's AWS platform, the original MD5_1 will be sent by the owner when uploading, and a re-computed MD5_2 is sent by AWS when downloading. In contrast, to provide data integrity, Microsoft's Azure Storage Service stores uploaded data MD5_1 in database and returns it to the user when it wants to retrieve data [30]

This procedure is secure in each individual session. Integrity and confidentiality of data during transmission are guaranteed by SSL protocol. However, from the perspective of Cloud storage, data security depends not only on uploading and downloading phases, but also on the maintenance phase. The Uploading phase ensures that data received by Cloud providers is the same as the owner uploads. The downloading phase guarantees that what the user retrieves is the same data the Cloud provider sends. Unfortunately, there is one critical link missing that is required to protect or track data stored in Cloud storage.

2.2 Secure Cloud Storage Architectures

Since Cloud storage is a service delivered over the Internet and hardware and system software provide the service, traditional distributed storage can be looked at as a specific case of Cloud storage. Security problems in distributed storage systems, such as authentication, authorization, availability, confidentiality, integrity, key sharing and management, auditing and intrusion detection, usability and performance, should also be considered in Cloud storage.

Kher and Kim [17] presented a survey of existing secure storage systems and listed some solutions. However, Cloud storage systems have their own features. The Cloud Security Alliance's report [31] lists 15 different issues and Chow *et al* [7] group them into three categories: traditional, availability and third-party control. Cachin *et al* [4] presented a brief survey of solutions to secure Cloud storage. They use a "provable data possession" (PDP) [1] model or a "proof of retrievability" (POR) [13] model for ensuring possession of a file during a maintenance phase. Such models and their derivatives can efficiently and sufficiently find gross omissions such as 1% data loss and are effective. They also use some protocols, such as SUNDR [18] to realize fork-consistent storage.

Based on recent proposed non-standard approaches, Kamara and Lauter [15] described a secure Cloud storage architecture. They believe "*confidentiality, integrity, availability, reliability, efficient retrieval, data sharing*" services should be provided. In their architecture, there are four components: a token generator (TG) to generate indices that enable the provider to search data, a data processor (DP) to encrypt the data by some methods, such

as AES, searchable encryption, a data credential generator (CG) for an access control policy, and a data verifier (DV) to check integrity. In their architecture, Cloud storage providers are responsible for availability and reliability. DP is responsible for confidentiality. DV is responsible for integrity. CG is responsible for data sharing. TG is responsible for data retrieval. POR/PDP is responsible for proof of storage.

Kamara's work is useful for proving the integrity of data, but it is not perfect for a holistic Cloud solution. Raluca Ada Popa *et al* [25] presented architecture for secure Cloud storage. They divided the security properties of Cloud storage in four categories: confidentiality, integrity, write-serializability and read freshness. With signed messages and chain hash, the architecture can provide non-reputable and write-serializability property. Freshness is guaranteed by periodically auditing data.

2.3 Security Issues for Cloud Storage

Obviously current Cloud storage platforms can meet basic requirements of mass storage at a low cost. We can also enhance the security of Cloud storage using some of the methods mentioned above. Confidentiality and Integrity are achieved through robust encryption and Message-Digest respectively. Non-repudiation is provided by the exchange of signed message-digests [8] Freshness is guaranteed by periodic audit. Write-serializability is supported by chain hash or persistent authenticated dictionary (PAD) [10] . SUNDR can be used to defend "fork consistency attack". Broadcast encryption [3] and Key rotation [14] are used to improve scalability.

It seems that security problems in Cloud storage systems are covered. Nevertheless, one of the crucial aspects of Cloud storage is that none of the three entities should always be trusted. Any one of them could be malicious. That is why non-repudiation is a key mechanism [25] in secure Cloud storages.

However, although signed digests are used for non-repudiation in Cloud storage [8] [25] , they did not consider "Fairness" too much. Fairness means that

"*no party gains an advantage over another at any moment during the running of the protocol. The protocol would not be fair, for example, if one of the parties obtained the signed contract without the other being able to do likewise*" [21] .

It is obvious that "Non-repudiation" in references [8] and [25] is not "fair" since any party can refuse to send his own certification after it receives the sender's certification. Then non-repudiation method in references [8] and [25] is just the case 1 in [28] . This will lead to two problems:

1. *Space Consistency*: How can a user ensure that the received data are the same as that the owner has uploaded earlier? Has it been tampered when stored in the cloud?

2. *Fairness*: If a sender does not get any response from its peers, what can it do once disputation happens?

Although PAD can be used to solve the consistency problem, it has an assumption that the users need to know the element in Cloud. SUNDR can also be used in the consistency problem. However, it still has two assumptions. At least one user has the correct updated data, and other users can communicate with the user and the user should be trusted.

References [8] [9] [25] proposed a prototype. However, more details should be considered. Otherwise, such non-repudiation protocol could be easily attacked [19] . Thus we propose a novel fair multi-party non-repudiation (MPNR) protocol for Cloud storage.

3. A NEW MPNR PROTOCOL

3.1 Notation and Definitions

For a description of our fair non-repudiation protocol, some notation and definitions are used as follows:

NRO: Non-Repudiation of Origin, which is held by the recipient and is intended to protect against the sender's false denial of having originated the message by being presented to an arbitrator [27] who can unambiguously decide whether the sender is the author of a given message or not.

NRR: Non-Repudiation of Receipt, which is held by sender and intended to protect against the recipient's false denial of having received message by being presented to an arbitrator [28], who can unambiguously decide whether the recipient received a given message or not.

Timeliness: This is achieved if and only if all honest parties always have the ability to reach, in a finite amount of time, a point in the protocol where they can stop the protocol while preserving fairness. Timeliness avoids situations where a party does not know whether it can stop the protocol without loosing fairness or not. A multi-party protocol is said to respect timeliness if all honest entities are able to terminate the protocol in a finite amount of time without losing fairness.

L: Unique Label throughout the session. Here we suppose L is the hash of the data and the entities.

Flag: Indicate the purpose of the step.

$EG_B\{\}$: any group encryption scheme that only recipients $Bi \in B$ can decrypt it. NRO_{OU} is encrypted with group encryption scheme. We do not imply that is the only option. Different schemes can be used under different scenarios. For example, if there are fewer recipients, we can encrypt the NRO_{OU} with each recipient's public key, add a label for location and concatenate them together. If there are many recipients, we can use other group encryption methods [6] . When user B_i downloads the data from the Cloud, it will decipher it using K, where K is the key for B to decipher the data.

$E_X(M)$: asymmetric encryption of message M with party X's public key.

$S_X(M)$: signature of the message by party X, normally with X private Key.

H(M): one-way hash function over the message M.

X→Y: party X sends a message M to party Y.

X↔Y: party X fetches a message M from party Y.

3.2 The Fair MPNR Protocol

This section presents a new MPNR protocol. To date, there are only two approaches for fair non-repudiation [14], [16], [20], [26]. One is gradual exchange, which is not practical. Another is Trusted Third Party (TTP), which is used in nearly all non-repudiation protocols. In traditional non-repudiation applications, four steps are required to finish the fair non-repudiation protocol.

#1. A sender sends an encrypted message $E_K(M)$ to a recipient with an NRO.

#2. The recipient responds with an NRR.

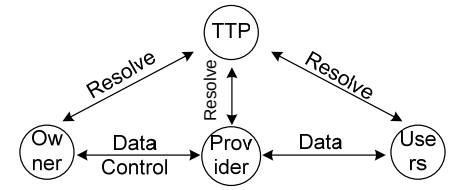
#3. After the sender gets an NRR, it will send the key to the recipient with an NRO.

#4. The recipient responds with an NRR.

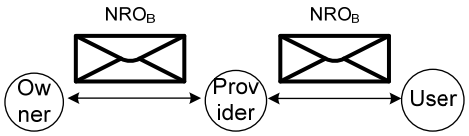
#5. An entity can initiate Resolve mode if it is needed.

In this paper, owner and users are senders and we denote them "A" and "B", respectively, and Cloud provider as recipient is denoted "C" The architecture is illustrated in Figure 3. In the new MPNR protocol, we still use a TTP to guarantee "Fairness" as illustrated in Figure 2(a). In addition to normal upload/download processes, Resolve mode guarantees that every party is able to complete or abort the execution of protocol, without being forced to wait for responses from other parties, who are potentially malicious or irresponsible.

We assume communication channels between the peers and TTP are resilient and reliable. "Resilient" means messages will be eventually received. We also assume that generally all parties are willing to complete the transactions by themselves and the TTP is only required as a last resort. None of the parties acts against its own interests. For "Consistency", the key idea is that in our MPNR protocol the provider is just like a hollow man-in-the-middle. The rationale is to bridge the uploading and downloading data with a fair non-repudiation receipt through group encryption. Data from the owner to users is packed and encrypted. The provider can only check the integrity of packed data and decide whether users can access data according to an access list managed by the owner. The provider cannot know the content of data and cannot tamper data, as in Figure 2(b).



(a). Fairness.



(b). Consistency.

Figure 2. MPNR for Cloud Storage.

The MPNR has two modes: Normal mode and Resolve mode. Normal mode needs two rounds: Uploading session between Owner and Provider and Downloading session between Users and Provider. Normal mode is similar to that in references [8] [25] It supposes the two peers are willing to exchange messages and non-repudiation evidence, and messages cannot be lost during transmission. If the sender failed to obtain non-repudiation evidence, it can invoke a Resolve mode through the TTP.

3.2.1 Normal Mode

A. Uploading Session: Owner <=> Provider

A sender uploads data to Cloud and updates it later.

In this step, Owner encrypts data with key K, generates two proofs of non-repudiation: NRO_{OU} and NRO_{OC} for the users and the Cloud provider respectively, where "OU" means the evidence is produced by the owner and will be stored by the user. "OC" has the similar meaning. NRO_{OU} is critical for multi-party communication. K is the key used to decipher the data. Users will verify the data integrity $S_A(H(DataToBeStored))$ after they download data. The sender uses the group encryption scheme to guarantee that only the recipients in the B_list can

decipher the $NRO_{OU.}$ This signed hash is very critical since it makes up the missing link between the uploading and downloading sessions. Owner encrypts the NRO_{OC} with the provider's public key and delivers the NRO_{OC} to the Cloud as the non-repudiation evidence. When the owner wants to update data or the user access list or abort, it just needs to modify the content of Request. The step is described as follows:

Step 1**:** *A => C;*
Request = {L, A, C, TTP, DataToBeStored, B_List, H(DataToBeStored), H(B_List), Seq, flag, Tg, T1, $EG_B\{NRO_{OU}\}$, $E_C\{ NRO_{OC}\}$*}*

Where:
NRO_{OU} := {K, L, S_A(H(DataToBeStored))}, it is open to recipients but it is not open to Cloud service provider.

NRO_{OC} := {S_A(H(DataToBeStored), H(B), $EG_B\{NRO_{OU}\}$, H(L, Seq1, f1, Tg, T1)}. NRO_{OC} is the evidence of non-repudiation.

Seq1: represents the unique sequence number in each message. Each step must be unique throughout the whole transaction in order to prevent reply attacks.

Tg: The time when the message is generated. It should be authenticated since all other timestamps are generated based on it.

T1: The time limit for sender to wait for the NRR from recipient in Step 1.

B_List, H(B_List): the list of authenticated recipients who can download and decrypt the data and its hash value.

Step 2**:** *C => A:*
Response = {L, A, C, TTP, H(DataReceived), H(B), Seq2, flag, Tg, T2, Ts, $E_A\{NRR_{CO}\}$*}.*

Where:
NRR : = S_C(H(DataReceived)), S_C{H(L, Seq, flag, Tg, T2, Ts, NRO_{OC})}.

Ts: The time when data is stored, it provides sender the evidence of data storage time.

Once the message is received from the owner in Step 1, the Cloud storage service provider verifies the validity of L with A, C, and TTP, H. If valid, the service provider decrypts message with its private key and conducts further verification of the integrity of parameters in the Request. Then, the integrity of data will be checked. When all procedures are completed without any anomalies detected, the service provider sends NRR to sender before time is up. Otherwise, the service provider will respond with an ERROR message. On receiving the NRR_{CO}, the owner verifies whether the hash of data H(DataReceived) is the same as what it has sent and validates the NRR. Then, the owner stores NRR for future. Otherwise, it initiates Resolve process.

After the provider sends an NRR to the owner, there are two possibilities. One is that the message is lost and the owner cannot get the NRR_{CO}. In such case, provider will get a request from TTP through Resolve mode. Otherwise, it means that owner gets and agrees to NRR_{CO}. The uploading session ends. T2 should not be shorter than T1.

B. Downloading Session: Users <=> Provider

When any user wants to download data, it should send a request with non-repudiation evidence NRO_{UC} to the Cloud provider. The request includes the user's identity. The provider will validate the request and verify whether the user is in the list of B that was previously sent by the sender. If it is, the data along with $EG_B\{NRO_{OU}\}$ will be sent to the user with provider's non-repudiation evidence.

When the user gets data and EG_B, it will obtain K and H(Data) by decrypting $EG_B\{NRO_{OU}\}$ and check the integrity. Meanwhile, the user will also check the validity of the NRR_{CU}. After the provider responds NRR_{UC} to users, there are two possibilities. One is that message is lost and the owner cannot get NRR_{UC}. In such case, provider will get a request from TTP through Resolve mode. Otherwise, it means that the owner gets and agrees to the NRR_{CU}. The downloading session ends. T4 should not be shorter than T3.

Now, the hash value H(DataToBeStored) between the owner and the users is exchanged through the Cloud provider without requiring extra channels, and it is hidden from Cloud provider. We use H(DataToBeStored) as the integrity link to guarantee the "Consistency".

Step 1**:** *B => C;*
Request = Li, A, C, Bi, TTP, Seq3, flag, Tg, T3, $E_C\{NRO_{UC}\}$*.*

Where
NRO_{UC} : = S_{Bi}H(Li, A, C, TTP, Seq3, flag, Tg, T3)},
Li := H(A, C, Bi, TTP)

Step 2**:** C => Bi:
Response = {L, A, C, Bi, TTP, DataRetrieved, H(DataRetrived), Seq4, flag, T4, $EG_B\{NRO_{OU}\}$, $E_{Bi}\{NRR_{CU}\}$*}*

Where
NRR_{CU} := S_C(H(DataRetrieved), $S_C\{EG_B(NRO_{OU})\}$, S_C{H(L, Seq4, flag, T4)}, and T4 is the time limit.

3.2.2 Resolve Mode

Anomalies do not necessarily lead to the termination of a transaction. The owner or user needs fair non-repudiation evidence. Thus, a process of error correction and/or anomaly resolution is required in this protocol. The sender sends the message identification and evidence to TTP to start a recovery process. TTP will transfer the request to the provider with a time limit. If the provider agrees to continue the process, it will return a message to TTP with NRR before time out. Provider also should restart the lost step 2.

There are two possibilities for sender. One is that the recipient still refuses to respond. In such case, after time is out, TTP will generate evidence of NRR to sender. The Resolve session is done. Since the channel in Resolve mode is resilient and the TTP is trustable, the arbitrator would look at the NRR as the evidence. Another case is the recipient will respond the TTP and restart the step 2. After the sender receives NRR from the recipient, it will inform TTP and Resolve session is done. Figure 3 is the flowchart.

3.2.3 Resolution of Disputation

Having studied the behaviors of each party in the MPNR protocol, there are five typical possible disputations as summarized as follows:

Case 1: Owner and users collude to blackmail the service provider. The owner stores some data in the Cloud first; then a user downloads the data. They claim that the data has been tampered and ask the service provider to pay the so-called loss. The service provider can easily prove his innocence by presenting the NRO_{OC} and NRO_{UC}.

Case 2: The service provider has received the NRO but does not respond with a NRR in order to have some advantage over the senders. The senders can handle such type of activity by initiating a Resolve mode through the TTP. The service provider cannot get any advantage.

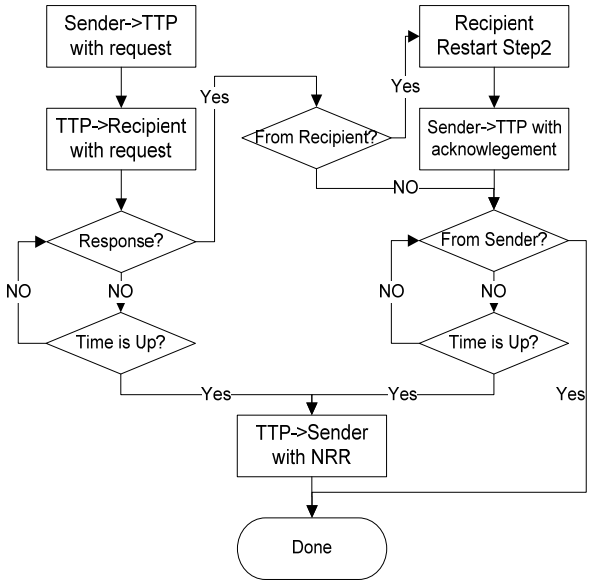


Figure 3. Flowchart of Resolve mode.

Case 3: Consider the scenario where the provider wants to charge more service fees. When owner stores 500GB data, but provider claims that it stored 1TB. The owner can verify the earlier transaction through NRR_{CO}, and NRO_{OC} would not support the provider. Similarly, this evidence also helps the provider when the users try to deny the service fees.

Case 4: There is the possibility that disputations happen between the owner and users. The owner can claim that a user has received the data but the user denies. The arbitrator can easily figure out the truth by the NRO and NRR.

Case5: Our MPNR protocol is also helpful to deny access to unauthorized users. For instance, a client Bob claims that it has permission to access the data. This can be easily verified by checking the list of recipients with NRO_{OC}.

3.3 Performance Discussion

The proposed non-repudiation protocol is like the TCP/IP three-phase handshaking protocol. It is designed to exchange evidence in the data transaction, which removes ambiguities that lead to repudiations or disputations among users or between a user and service provider. Actually, all of the papers about on-repudiation focus on the protocol design, and there is no experiment for performance reported [1] ,[5] ,[16] ,[21] , [24] ,[26] .

Moreover, Cloud storage is different from the traditional distributed storage. In the traditional distributed storage applications, data is normally exchanged through the Internet. For the cloud storage, however, since the data is often large in size, such as GB or TB volume, the bottleneck definitely lies on the uploading and downloading process.

Therefore, current cloud storage services, such as S3, normally use the ship-method (e.g. FedEx) to solve the problem. Considered as the overhead, the time required for executing the protocol is very small compared to the time consumed by data transfer. For example, the time to compute the MD5 and SHA1 for 2GB is about 9s and 11s, but if we upload and download 2GB by internet, it can take up to half an hour or more.

If we consider small data such as 20MB, the overhead is less than 512 Bytes, (< 128 bytes for MD5 or < 384 bytes

for SHA-512, occupy 1 sector for disk), the latency time should less than 13ms (ST32000641AS hard disk, 2TB, 7200rpms, 64MB cache, seek time ~8.5ms, latency time ~ 4.2ms).

On the same computing power (Intel E8400, 3G RAM), considering the data transmission, the computational time depends on the data size. MD5/SHA-512 needs ~9s/12s for 2GB bytes because of different rounds (64/80 iterative steps). AES encryption for 2GB needs approximately 180s. The signature generation time is about 100ms (ECDSA, ARM7TDMI, 50MHz) [32] . The total (needs 3 DSA) is about 300ms.

Data transmission time, which normally takes days or hours, is by far much longer than time required by other operations. For example, consider a data set of 2GB, the time for encryption is about 180s, time for Digital Digest is merely 12s, time for signature generation is only about 300ms, and the latency time is about 13ms. Obviously, days/hours >> 180s >> 12s >> 300ms >> 13ms.

Compared to shipping time, the protocol execution time is relatively trivial. Additionally, there are various factors that influence the performances including disk type, system architecture, algorithm, etc. Thus, we leave the experimental study of performance evaluation as our next step work considering the complexity.

Additionally, the entities still need similar confidentiality and integrity operations even without the MPNR. The MPNR protocol adds a very small extra burden for each entity.

4. SECURITY ANALYSIS

The goal of non-repudiation is to enhance the security of Cloud storage and convince potential customers that the service is secure. Therefore, it is highly desired that the MPNR protocol is robust against various threats. In this section, we analyze the robustness of MPNR protocol under some general malicious attacks and some specific attacks for non-repudiation protocols.

4.1 General Attacks

Man-in-the-middle attack

The Man-in-the-middle attack (MITM) [23] 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 being exchanged between the two victims and inject new ones. However, an MITM attack can succeed only when the attacker can impersonate the end party. It can be prevented by authentication. In our MPNR protocol, authentication and digital signature are required for the purpose of eliminating disputation. Automatically, when the parties get the other's public key, they should authenticate the validity against the MITM.

Reflection Attacks

A reflection attack [23] is a method that attacks a challenge-response authentication system that uses the same protocol in both directions. The protocol proposed in this paper is not a challenge-response authentication system. Furthermore, each message contains a unique identifier, and thus the reflection attack can be avoided.

Interleaving Attacks

The interleaving attack [23] is similar to the 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, and each session is finished only in one round. Therefore, the interleaving attack cannot threaten the MPNR protocol.

Replay Attacks

Replay attack [23] is a network attack in which a valid data transmission is maliciously or fraudulently repeated or delayed. This is carried out either by the sender or by an adversary who intercepts the data and retransmits it.

The replay attack can be defended by the use of challenge-response techniques and by embedding the target ID party in the response or the timestamp. In our protocol, we use unique sequence number with the sender signature to avoid the attack. For example, an adversary Eve has intercepted the message and replayed it to TTP. Even though it can modify the SeqN in the plain text; the hash value that has been encrypted by the sender's private key cannot be tampered without being detected.

4.2 Specific Attack

Timeline attacks

Timeline attacks are typical in non-repudiation protocols. In fairness, each party can stop the execution after a pre-fixed time out. In this protocol, the Tx field is used in each message to limit the reception time of a message. Thus, when a party receives a message, it will check the validity of the Tx with the actual time. If it is invalid, the party discards the message and initiates the resolve mode. However, simply grafting some note of expiry may also cause trouble. Consider the following protocol [28] :

Step	Acts	Parameters
1	A=>B	B,L,T,C,NRO
2	B=>A	A,L,NRR

3	**A=>TTP**	**B,L,T,K,sub_K**
4	**A<=>TTP**	**A,B,L,T0,K,con_K**
5	**B<=>TTP**	**A,B,L,T0,K,con_K**

Step 4 and step 5 can be conducted concurrently. Since T is the time limit on the TTP's clock and T0 is the time that the confirmed key has made available to the public, it remains so until time T. However, party A can delay step 3 up to the last moment before T, so that it can perform step 4 while standing a good chance that B might subsequently miss step 5 [19] .

Another example is shown below by adding a time limit. Where *B* adds a time limit T1 in step 2, T1 < T. *B* wants *A* to perform step3 before T1 to avoid the problem above. When there is repudiation, the adjudicator checks that T0< T1< T. But, since TTP does not know T1, *B* can give time limit T1 < T0. After *B* gets K and the decrypted message, it claims that the protocol execution is invalid [16] .

Step	Acts	Parameters
1	**A=>B**	**B,L,T,C,NRO**
2	**B=>A**	**A,L,T1, NRR**
3	**A=>TTP**	**B,L,T,K,sub_K**
4	**A<=>TTP**	**A,B,L,T0,K,con_K**
5	**B<=>TTP**	**A,B,L,T0,K,con_K**

In our MPNR protocol, this attack is not possible since there is only one round in one session, and no one can get an advantage over another. Additionally, in each step, the party tracks the time limit clearly.

Reuse of ETTP(K)

In one non-repudiation protocol [20] , sub_K contains only items sent as parts of the first message. In particular it contains E_{TTP} (K). Thus *B* can reuse E_{TTP} (K) in a different protocol run with *B'* and produce a valid sub'_K that consists of $S_B(f_{sub}, B', L', E_{TTP}(K))$, where L' is a new random label.

By using this sub'_K together with appropriate EOO'_C and EOR'_C values in the resolve sub-protocol, *B* gains K and thus learns the message M. *A* cannot receive any evidence of receipt for this message, as A has only enough information to run the abort sub-protocol.

But as *B* executes the resolve sub-protocol under a different label L', the attack always succeeds. Thus, the protocol is unfair for *A* (assuming that knowledge of M is valuable information for *B*) [11] . This type of attack would have no impact on our MPNR protocol since there is no need to generate a valid sub_K.

Reuse of Labels and Keys

In some protocols, labels are equivalent to H(Data, K) and are unique. However, *B* cannot know the Data until the last step. This property implies that *B* can only check the validity of L in the last step. Under certain situations, TTP also cannot check the validity of L since TTP never gets the message for confidence.

For example, *A* can initiate the protocol with data M' but using the wrong label L = H(M, K). *B* cannot verify its validity until the last step. Therefore A can receive evidence of receipt for K from *B* if *B* forgets to check the label or from TTP since TTP cannot check the label at all. When *B* detects the error and initiates the resolve process, TTP may reject its request since *A* has aborted the transaction already. Such an attack cannot threaten the MPNR protocol since each party can check the validity of the Label in every step. TTP can also check the validity of the Label of each step.

Wrong sub_k Attacks

This attack is special to certain NR protocols. Let's consider the NR protocol proposed in. If *A* sends a wrong $E_{TTP}(K)$, the resolve protocol has to stop with an error when it is initiated by *B*. Then it prevents *B* from terminating the transaction. However, *A* can construct a resolve request with the correct encryption of the key and then *A* can complete the protocol at any time.

In our MPNR protocol, the attack is not feasible since TTP only checks the consistency in the resolve mode and it is the responsibility of *A* and *B* to decide the result of the resolve procedure. If the sender sends a wrong message, the message cannot reach the other party and the sender cannot take any advantage.

5. CONCLUSIONS

This paper reports our work on data security in Cloud storage. We have revealed the existing vulnerability in Cloud storage due to the missing connection between the robust uploading and downloading phases. We proposed a new MPNR protocol that is specifically designed for the Cloud storage environment. This protocol can enhance the security of Cloud storage and make it more reliable for potential consumers.

REFERENCES

[1] B. Agreiter, M. Hafner, and R. Breu, "A Fair Non-repudiation Service in a Web Service Peer-to-Peer Environment," Computer Standards & Interfaces, vol 30, no 6, pp372--378, August 2008.

[2] Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song. "Provable data possession at untrusted stores". In Proc. *ACM CCS 2007*, 2007.

[3] D. Boneh, C. Gentry, and B. Waters, "Collusion Resistant Broadcast Encryption with Short Ciphertexts and Private Keys," *Lecture Notes in Computer Science*, 2005.

[4] C.Cachin, I.Keidar, and A. Shraer, “Trusting the Cloud”, *ACM SIGACT News*, 20:4 (2009), pp. 81-86.

[5] M. Carbonell, J. M. Sierra, and J. Lopez, “Secure Multi-Party Payment with an Intermediary Entity,” *Computers & Security*, vol 28, no 5, pp289--300, July 2009.

[6] G. Chiou and W. Chen, “Secure broadcasting using the secure lock,” *IEEE Transactions on Software Engineering*, vol 15, no 8, pp929-934, Aug. 1989.

[7] R.Chow, P.Golle, M.Jakobsson, E.Shi, J.Staddon, R.Masuoka, and J.Molina, “Controlling data in the Cloud: outsourcing computation without outsourcing control”, Proceedings of *CCSW 2009*, November 13; Chicago, IL. NY: ACM; 2009; 85-90.

[8] J. Feng, Y. Chen, and P. Liu, “Bridging the Missing Link of Cloud Data Storage Security in AWS”, the *2010 IEEE* CCNC’10, Las Vegas, Nevada, USA, January 9 - 12, 2010.

[9] J. Feng, Y. Chen, W.-S. Ku, and P. Liu, "Analysis of Integrity Vulnerabilities and a Non-repudiation Protocol for Cloud Data Storage Platforms," the *2nd International Workshop on Security in Cloud Computing*, in conjunction with ICPP 2010, San Diego, CA, USA, Sept. 14, 2010.

[10] A. Anagnostopoulos, M.T. Goodrich, and R. Tamassia, “ Persistent authenticated dictionaries and their applications”, In *International Conference on Information Security* (ISC), Seoul, Korea, Dec. 2001.

[11] S. Gurgens, C. Rudolph, and H. Vogt, “On the Security of Fair Non-repudiation Protocols,” *Proceedings of 2003 Information Security Conference*, Bristol, UK, October 2003.

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

[13] A. Juels and B. S. K. Jr. Pors: “Proofs of retrievability for large files”. In *Proc. ACM CCS*, pages 584–597, 2007

[14] M. Kallahalla, E. Riedel, R. Swaminathan, Q. Wang, and K. Fu, “Plutus: Scalable Secure File Sharing on Untrusted Storage,” In *USENIX FAST*, pages 29 ~ 42, 2003.

[15] S. Kamara and K. Lauter. “Cryptographic Cloud storage”, In *ACM Workshop on Cloud Security*, 2009

[16] K. Kim, S. Park, and J. Baek, “Improving Fairness and Privacy of Zhou-Gollmann's Fair Non-repudiation Protocol,” *Proceedings of 1999 ICPP Workshop on Security*, pp 140-145, Aizu, Japan, September 1999.

[17] V. Kher and Y. Kim, “Securing distributed storage: challenges, techniques, and systems”, Proceedings of *the 2005 ACM workshop on Storage* (StorageSS'05), pp9-25, Fairfax, Virginia, USA, 2005.

[18] J. Li, M. Krohn, D. Mazieres, and D. Shasha. “SUNDR: Secure untrusted data repository”, *In OSDI*, 2004.

[19] P. Louridas, “Some Guidelines for Non-repudiation Protocols,” *Computer Communication Review*, vol 30, no 5, October 2000.

[20] O. Markowitch and S. Kremer, “A Multi-Party Optimistic Non-repudiation Protocol,” *Proceedings of 2000 International Conference on Information Security and Cryptology*, pp 109--122, Seoul, Korea, December 2000.

[21] A. Ruiz-Martínez, C.I. Marín-López, L. Baño-López and A. F. Gómez-Skarmeta, “A New Fair Non-repudiation Protocol for Secure Negotiation and Contract Signing”, *Journal of Universal Computer Science*, Vol.15, No.3, February , pp 555-584, 2009

[22] T. Mather, S. Kumaraswamy, and S. LatifCloud, “Security & Privacy”, O’Reilly, 2009,

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

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

[25] R. Popa, J. Lorch, D. Molnar, H.Wang, and L. Zhuang, “Enabling Security in Cloud Storage SLAs with CloudProof”, *Microsoft TechReport MSR-TR-2010-46*, May, 2010.

[26] A. Ruiz-Martinez, I. Marin-Lopez, L. Bano-Lopez, and A. F. Gomez-Skarmeta, “A New Fair Non-repudiation Protocol for Secure Negotiation and Contract Signing,” *Journal of Universal Computer Science*, vol 15, no 3, February 2009.

[27] J. Zhou and D. Gollmann, “A Fair Non-repudiation Protocol,” *Proceedings of 1996 IEEE Symposium on Security and Privacy*, pp 55--61, Oakland, USA, May 1996.

[28] J. Zhou and D. Gollmann, “An Efficient Non-repudiation Protocol,” *Proceedings of The 10th Computer Security Foundations Workshop. IEEE Computer*, pp 126-132, Oakland, USA, May 1996.

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

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

[31] Security Guidance for Critical Areas of Focus in Cloud Computing, http://www.cloudsecurityalliance.org /guidance /csaguide.pdf.

[32] http://www.certicom.com/index.php/software-security-solutions

[33] Privacy in the Cloud Computing Era, http://download. microsoft.com/download/3/9/1/3912E37E-5D7A-4775-B677-B7C2BAF10807/cloud_privacy_wp_102809.pdf, Nov, 2010.