How to improve monitoring and auditing security properties in cloud storage?

Carlos André Batista de Carvalho[1,2], **Nazim Agoulmine**[3], **Miguel Franklin de Castro**[1], **Rossana Maria de Castro Andrade**[1,*]

[1]Graduate Program in Computer Science (MDCC),
Group of Computer Networks, Software Engineering, and Systems (GREat),
Federal University of Ceará (UFC)

[2]Computer Science Department, Federal University of Piauí (UFPI)

[3]IBISC Laboratory, University of Evry (UEVE)

candrebc@ufpi.edu.br, nazim.agoulmine@ufrst.univ-evry.fr,

{miguel, rossana}@ufc.br

Abstract. *A cloud storage service implements security mechanisms to protect users' data. Moreover, due to the loss of control over the cloud infrastructure, auditing and monitoring mechanisms are used to detect violations of security properties, increasing the trust and transparency in cloud services. However, there are flaws in existing solutions to ensure integrity, freshness and write-serializability properties. Then, we propose a monitoring and auditing mechanism to verify these properties, allowing to detect violations that are not identified by other solutions. Colored Petri Nets (CPNs) are used to model and validate the proposed mechanism. As results, the provider cannot deny the detected violations, and attacks are detected in real-time, except collusion attacks, identified only in our auditing phase.*

1. Introduction

Cloud computing is a distributed computing paradigm that enables the sharing of computational resources between many clients. It is possible to reduce the infrastructure costs by contracting a public cloud provider and paying only for the consumed resources. Besides, the services' scalability allows the dynamic allocation of resources, in accordance with customers' needs. However, this technology comes with the main drawback of losing control over the cloud infrastructure. Therefore, individuals, companies and organizations are resisting to adopt public clouds, due to concerns about security and privacy [Luna et al. 2015].

Cloud providers have developed security mechanisms based on frameworks and security guidelines elaborated by standardization bodies, such as ISO (International Organization for Standardization), NIST (National Institute of Standards and Technology) and CSA (Cloud Security Alliance) [Luna et al. 2015]. However, their customers have a limited view of the security, requesting more transparency with mechanisms that provide service security guarantees [Luna et al. 2015]. In this context, scientific

[*] Researcher Scholarship - DT Level 2, sponsored by CNPq

research has been realized to develop solutions that improve the trust in cloud providers [Bamiah et al. 2014]. It is important to highlight the audit and monitoring mechanisms to prove the security and allow the violation detection [Popa et al. 2011, Hwang et al. 2014a, Tiwari and Gangadharan 2015]. These mechanisms can be used by a solution of Service Level Agreement (SLA) management to specify and ensure security properties [Carvalho et al. 2017a].

Among security concerns (*e.g.*, data loss and leakage, resource location, service disruption and multi-tenancy issues), [Rong et al. 2013] stress that the big concern is the assurance of the security in cloud storage. In this context, it is important to prove that security properties are achieved. Usually, confidentiality, integrity and availability are the required security properties [Zou et al. 2015], but the literature highlights other properties related to secure cloud storage, such as retrievability [Tiwari and Gangadharan 2015], freshness [Popa et al. 2011], write-serializability [Popa et al. 2011] and location [Albeshri et al. 2014]. The use of cryptography is fundamental to offer security, and to prevent, for example, data leakage. However, it is not possible to avoid all kind of attacks, and the service monitoring is essential to identify an attempted attack and to block it.

Normally, the existing solutions are based on audit mechanisms to detect if security properties were violated. The problem with this approach is that violations are detected after an attack, only when the audit is performed. Although the provider may be penalized, maybe it is not possible to recover the damage. Besides, there is the cost of the audit mechanisms that introduce an overhead in the system, and are performed by a Third-Party Auditor (TPA). Then, it is interesting the development of mechanisms to detect violations in real-time, allowing to avoid an attack or to reduce the damage [Hwang et al. 2014a].

Besides, in some solutions, the security assessment is not formally presented, resulting in security flaws. For example, in our preliminary study [Carvalho et al. 2016], the Cloudproof [Popa et al. 2011] was modeled and evaluated using Colored Petri Nets (CPNs). As a result, it was possible to identify scenarios in which security violations were not detected by Cloudproof. Therefore, the design of mechanisms to guarantee security properties is still challenging.

In this paper, we propose a mechanism to verify integrity, freshness, and write-serializability, allowing to detect violations that were not identified by existing solutions. The provider cannot deceive this mechanism, denying a detected violation. In addition, the proposed mechanism will verify security properties in real-time. However, we identify an attack in which the audit is still necessary to detect it. In order to demonstrate the security and guarantee these properties, we model and validate the proposed mechanism using CPNs.

The rest of this paper is organized as follows. In Sections 2 and 3, we discuss, respectively, the background related to secure cloud storage and the related work. The proposed mechanism is described in Section 4 and evaluated in Section 5. Lastly, Section 6 presents the final considerations and future work suggestions.

2. Secure Cloud Storage

In this section, we show an example of a storage service that combines security mechanisms to offer security, transparency and trust in this service. We also describe the security properties and the threats inherent to our monitoring and auditing mechanism.

2.1. An example

A cloud storage service must, at least, allow users to create, read, update and delete files. The security of the stored data and file sharing are common requirements of this type of service. Then, the secure storage service has to include an access control mechanism, allowing the definition of permissions to each file. Figure 1 presents an overview and the stakeholders of this service in which a cloud customer can purchase this storage service, and define permissions for cloud users to perform cloud transactions. The cloud transactions are performed by a cloud provider and managed by a broker, and the security properties monitored by the users and audited by a Third-Party Auditor (TPA).

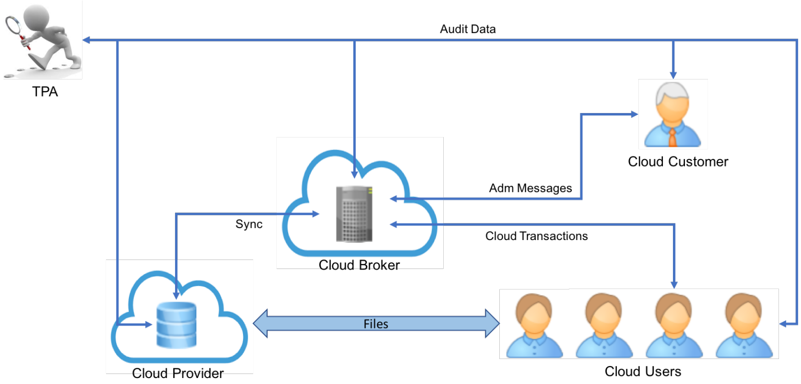

Figure 1. Storage service overview

An access control mechanism protects the service against unauthorized reading and data modification. Besides, a cloud customer can also change files permissions, granting or revoking privileges to users. However, access control and other security solutions are not analyzed in this paper, focusing in verification of security properties. Thus, we assume that the access control mechanism is secure so that only authorized users can perform cloud transactions.

In monitoring, a user analyzes their logs and the messages exchanged during a cloud transaction to verify security properties. When a violation is detected, a TPA can perform an audit to avoid that a user falsely accuses the cloud provider. An audit is also performed to detect violations that cannot be identified in real-time. In order to audit the service, the TPA receives the logs from the provider, broker and users. In addition, the involved entities must verify the messages sent through the service. Thus, it is possible to detect external attacks.

The broker and the cloud provider can be deployed in the same cloud infrastructure. However, when different infrastructures are used, it is possible to improve the security (*e.g.*, deploying the broker in a private cloud), and to store files in multiple providers. The use of multiple providers improves the service availability [Celesti et al. 2016], but this is out of the scope of our work.

2.2. Security Properties

Security mechanisms must be designed to protect a user against existing threats, holding the requested security properties. In this research, we analyze the integrity, freshness and write-serializability of the data stored in the cloud.

Integrity violations occur when data are modified or created by unauthorized entities, resulting in data corruption. Normally, the integrity verification is based on hash functions or messages authentication codes (MACs), and the security of these cryptographic primitives is attested by the scientific community. The freshness indicates the read of the updated file, and the write-serializability controls the writing order. So, the write-serializability ensures that the new version of a file overwrites the last version of it. The verification of freshness and write-serializability is inherent in the operations of, respectively, reading and writing. A writing can be performed to update a file or create a new one. [Popa et al. 2011] suggest writing an empty file to remove it. Due to the possibility of the provider denying a detected violation, it is necessary to sign all logs, providing non-repudiation of the cloud transactions [Hwang et al. 2014b].

The existing solutions analyze the logs to detect some violation. However, the provider can restore the stored files from a backup when some problem occurs, without modifying the logs. In this case, the provider can overwrite the old version of a file, without being detected in write-serializability verification [Carvalho et al. 2016]. Then, before updating a file, it is necessary to check if the current version of the stored file is the same as that indicated by the broker. In order to verify this, it is possible to read the previous file before each writing. However, this approach is not efficient, and the Proof of Retrievability (PoR) schemes can be used to verify the loss of a file, without recover all file [Tiwari and Gangadharan 2015, Albeshri et al. 2012].

Besides data corruption and loss, the data leakage is an important issue of cloud storage [CSA 2013]. Thus, the access control of the files should be only allowed for authorized users, providing confidentiality. Although the confidentiality is not addressed at this moment, a complete solution must include an access control mechanism. On the other hand, it is possible to occur a data leakage because the logs are accessed by the broker, provider and TPA. Then, these logs cannot have sensible information to preserve the users' privacy.

2.3. Threats and Attacks

The monitoring and auditing will increase the transparency of cloud storage services, enabling the detection and prove of violations. Thus, it is necessary to identify the attacks scenarios and evaluate if the proposed mechanism detect them. In this section, we describe the threats and attacks that can affect security properties. The focus is to identify the malicious acts of the provider, broker and external attackers.

In accordance with the Doley-Yao model [Dolev and Yao 1983], an external adversary can impersonate any entity (user, broker or provider), creating or modifying messages. However, a corruption attack is easily detected when the integrity is verified. Although an attacker cannot understand the content of a message, he/she can store it and send this message during a replay attack. Thus, all entities must include verifications to detect corruption or replay attacks. Besides the attackers, the broker and provider can have malicious behaviors, and the users must identify them.

A malicious provider can: i) send an outdated data; ii) write files out of order; iii) confirm a writing transaction, without committing it; and iv) perform a transaction of an unauthorized user. The last malicious behavior is addressed by the access control mechanism because unauthorized users do not have the credentials to read a file or create an uncorrupted file. A replay attack can result in writing files out of order. In a rollback attack, the provider restores the system to a previous state [Hwang et al. 2014b]. Consequently, a user receives an outdated file, or the files are written out of order, and the proposed mechanism detects the violation.

Likewise, a rollback in the broker results in the loss of the attestation of the last transaction. In this case, the provider can inform the stored attestations to prove that the problem occurs in the broker. It is important also to analyze the collusion attacks, where the broker helps the provider to deceive the violation detection mechanism. For example, during a rollback attack, the provider restores the system to a previous state and, the broker informs an old attestation in order to avoid the violation detection.

Besides, in another possible scenario, a broker and provider can restore the system to different states. In this case, if the broker restores to a state older than the provider, the violation in the provider could not be detected, reporting that the problem was only in the broker. Thus, an audit must use the information about last transactions of each user to identify the violation of the provider.

3. Related Work

Our analysis of literature reveals solutions that address security issues in cloud storage, especially related to data corruption, loss and leakage. Normally, the solutions combine security mechanisms to protect the system against several threats, holding security properties. In this section, we focus on analyzing the mechanism to verify security properties of the related work although other mechanisms are used.

For example, CloudProof combines an access control mechanism to provide confidentiality and a mechanism to verify integrity, freshness and write-serializability [Popa et al. 2011]. The cloud transactions are performed by CloudProof, using a protocol to get or put blocks in cloud storage, and attestations are used to record each read or write transaction. During each transaction, the integrity is verified, but violations of freshness and write-serializability are detected only in auditing, after sorting the attestations sent by the users. Due to the signature of the attestations, it is not possible to falsely deny or accuse the occurrence of a violation. The audit is periodically carried out, and some blocks can be randomly selected for auditing, reducing the overhead. Besides, the CloudProof can detect replay attacks.

Due to the informal discussion of the theorems presented by [Popa et al. 2011], we decided to use CPNs to model and validate CloudProof, in order to identify ambiguities and security flaws on it [Carvalho et al. 2016]. As results, we can highlight mainly the lack of management of concurrent requests by users and the identification of scenarios in which security violations were not detected. These scenarios exist because the detection of violations is based only on the chain of attestations, without considering the stored data. For example, a malicious provider can inform a writing attestation, without performing the writing in cloud storage. If the next transaction is a read, the violation can be detected, but this malicious behavior is not identified when a writing transaction is executed immediately after the first one. Then, it is necessary not only verify the chain of attestations but also if each write was rightly executed.

In another interesting study, [Hwang et al. 2014b] propose a protocol that detects rollback attacks although the attestations are stored by the untrusted provider. Besides, the authors propose a scheme to discard old attestations, and another to allow concurrent access. However, the above violation scenario is not detected by this mechanism nor by the proposal of [Hwang et al. 2014a].

On the other hand, this scenario does not occur in other solutions because the reading is always required before each writing [Albeshri et al. 2012, Tiwari and Gangadharan 2015, Jin et al. 2016]. Nevertheless, this approach is not efficient in a multi-user environment, because a file is blocked until a user finishes its reading, modifying and writing. Although the write-serializability is ensured, it is not informed that the previous reading is mandatory. In addition, the architectures, proposed in these papers, also include mechanisms to prove the retrievability, ensuring even scarcely accessed files were not lost.

The audit is mandatory to verify the retrievability, but freshness and write-serializability should be monitored by real-time mechanisms [Jin et al. 2016, Hwang et al. 2014a]. However, there is an error in the designed protocol by [Jin et al. 2016]. In this protocol, *"only the owner who holds the secret master key can rotate the root signing key to a new version"*, and, at the same time, the users rotate the **root signing key** when files are updated.

Besides, [Hwang et al. 2014a] specify the use of a trustworthy synchronization server to inform the users the current state of the provider. This assumption is acceptable when this server is deployed in a private cloud, and therefore the collusion attack was not analyzed. We studied the effect of the use of an untrusted broker and concluded that an audit is yet necessary (see Section 5). The audit can also be performed to solve any contestation due to the non-repudiation of the cloud transactions [Hwang et al. 2014b].

Table 1. Comparison of related work

Work	Security Properties			Real-time verification	Collusion attack
	Integrity	Freshness	Write-serializabilty		
Popa et al. 2011	Yes	Yes	Partially	Only integrity	Unfeasible
Hwang et al. 2014b	Yes	Yes	Partially	Only integrity	Unfeasible
Hwang et al. 2014a	Yes	Yes	Partially	Yes	Undetected
Albeshiri et al. 2012	Yes	Yes	Yes (inefficient)	Only integrity	Unfeasible
Tiwari and Gangadharan 2015a	Yes	Yes	Yes (inefficient)	Only integrity	Unfeasible
Jin et al. 2016	Yes	Yes	Yes (inefficient)	Fail*	Unfeasible

* The freshness verification depends on the knowledge of the lastest root signing key. However, the authors do not prove how the users obtain this key for real-time verification.

In Table 1, the related work are compared. It is important to highlight that the success of the write-serializability verification depends on the previous reading of a file. Besides, in some studies, the collusion attacks are not possible because there is no broker (or other third entity) participating of cloud transactions. However, the real-time detection of freshness violations is not also possible, making the auditing necessary to detect them.

Lastly, it is interesting to mention about the evaluation of related work. The trend in the literature is the use of standalone PCs to evaluate the cost of proposed algorithms together with an informal security discussion. The single exception is the Cloudproof that was deployed in the Azure infrastructure [Popa et al. 2011]. However, the performed experiments do not indicate the absence of security flaws.

4. Proposal

A secure storage service should include a mechanism to monitor and audit the security properties, enabling the violation detection. We propose a mechanism to verify integrity, freshness and write-serializability of the data stored in the cloud. This mechanism must be combined with an access control mechanism to provide a solution suitable for the file sharing environment. Existing work already combines these solutions, but limitations were found in the violation detection mechanism. Our mechanism fixes the found security flaws and enables the detection in real-time when a trustworthy broker is used. Even so, the audit is necessary to solve any contestation and to identify collusion attacks.

The files are ciphered to provide confidentiality, and an access control mechanism allows the client to define and update the files' permissions. Usually, in related work, the Access Control Lists (ACLs) are used to specify the permissions, and the key management is based on Broadcast Encryption and Key Rotation [Popa et al. 2011, Jin et al. 2016]. Thus, the users can obtain keys for reading and writing from file's metadata if they have permission. Although the access control is verified during the cloud transactions, we assume the correctness of its functioning and do not treat the procedures to grant and revoke permissions.

Thus, we can focus on the verification of security properties that is performed during reading and writing transactions. The broker manages the transactions, being responsible for the control of concurrent transactions and for sending of the last attestation. Using this attestation, a user verifies the integrity and freshness of the read file, or prepare a writing request that complies with the write-serializability.

The elements of an attestation are: **UserID**, **UserLSN**, **FileID**, **FileVersion**, **FileHash**, **TransactionType**, **ChainHash** and **Signatures**. The UserLSN represents the last sequence number used by each user, enabling to detect replay and rollback attacks. The FileVersion is essential to verify the freshness and write-serializability, and the FileHash to check the integrity. The type of the transaction (i.e., reading or writing) indicate whether the expect FileVersion must be replaced during an auditing. The ChainHash is used to build the chain of attestations and is computed over the data of the current attestation and the ChainHash of the previous one. Lastly, all involved entities (i.e., broker, user and provider) sign the attestation for non-repudiation purposes.

It is essential to specify the management of concurrent transaction. The broker must allow simultaneous readings of the same file, blocking only during a writing. This blockade is necessary to avoid the reading of the previous version of a file and the requests of writings with the same FileVersion. On the other hand, the concurrent access to different files is allowed because there is no possibility of violation resulting from such access.

The broker must store the last attestation of each file and send it to users to verify the security properties. Besides, the users store the attestations of their last transactions with each file and report a violation if a file version informed by the broker is older than the one stored by the user. It is mandatory the sending of all attestations of each user to identify other scenarios of collusion attacks because not all attacks are detected in real-time. Therefore, the proposed mechanism defines the building of one chain of attestation per file.

On the other hand, it is not necessary that each user manages one UserLSN per file so that a single UserLSN per user is enough to verify security properties. The broker and provider also store the current UserLSN of all users in order to detect replay attacks. With this approach, it is not possible to probabilistically choose the files to be audited as suggest by [Popa et al. 2011] because, if some file is ignored, some UserLSNs will not be found. Anyway, all files must be audited to detect all violations scenarios.

Before the audit, the provider must send all attestations to the TPA, and the broker and users send their last attestations. For each file, the TPA builds the chain of attestations, ordering them, and analyzes the sequence of the file's versions to prove that no violation occurs. Besides, it is also check the presence of all UserLSNs considering all chains of attestation. The attestations sent by users and broker are used to verify whether the provider hid some transaction. Otherwise, the TPA report a violation. Due to the signature of attestations, no entity can deny a violation. After the audit, the attestations can be discarded, except the last attestation of each file that will be chained with the attestations of the new epoch[1].

Thus, this mechanism includes auditing procedures to verify the assurance of security properties during cloud transactions. In addition, the monitoring of the properties is performed during the clouds transactions in order to detect violations and attacks in real-time. Due to space constraint, only the protocol to perform read transactions is detailed in Figure 2. Besides the attestations, the exchanged messages are signed, and a receiver verifies the authenticity of the sender. Thus, any entity can detect attacks when invalid signatures are used.

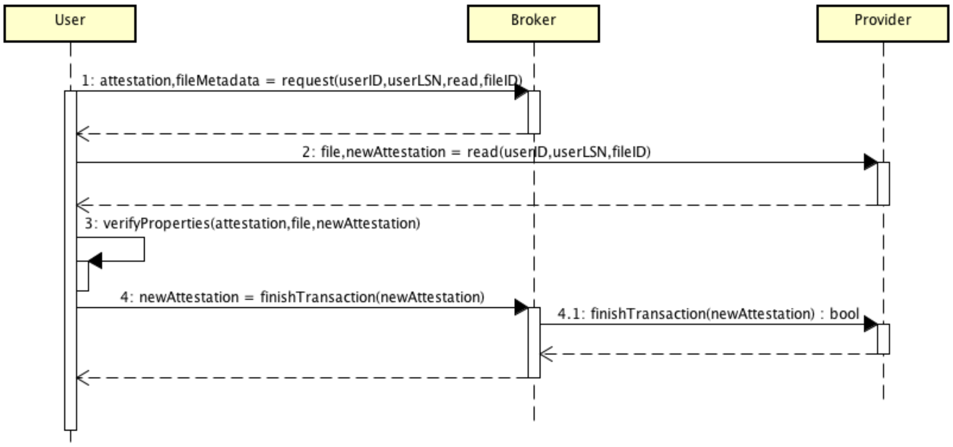
User
Broker
Provider
1: attestation,fileMetadata = request(userID,userLSN,read,fileID)
2: file,newAttestation = read(userID,userLSN,fileID)
3: verifyProperties(attestation,file,newAttestation)
4: newAttestation = finishTransaction(newAttestation)
4.1: finishTransaction(newAttestation) : bool

Figure 2. Reading a file

Before a user requests the reading of a file, a user sends a message to broker in order to receive the last attestation and the file's metadata. Next, this user deciphers the

[1] The period between two consecutive audits is called **epoch** [Popa et al. 2011]

encryption key and request the file to the provider. The provider sends the file and the new attestation signed by it. Then, the user verifies the integrity and freshness, using FileHash and FileVersion respectively. If no violation is found, the user and the broker sign the new attestation and send it to the provider. The broker and the user overwrite their last attestation.

During a writing, a user extracts the signature key to sign the new file. This signature is placed in the FileHash field of the attestation. The verification key is public, and the provider checks the signature, testing if the user is authorized before committing a writing. The provider also checks the FileVersion for write-serializability purposes.

One flaw of existing work is the verification of write-serializability based only on attestations. Thus, there are undetected violation scenarios, making it necessary the verification of the file version really stored by the provider. In order to check it, we use a PoR scheme, but its evaluation is outside the scope of this paper because we do not propose a PoR scheme. Besides, it was concluded, in related work, that the other option is more costly due to the full reading of the files [Albeshri et al. 2012, Tiwari and Gangadharan 2015]. In writing protocol, the PoR is performed before a writing so that a user makes a challenge and verifies the proof sent by the provider. The keys and tags used during this step are also stored in file's metadata. This metadata must be updated, with the tags related to the new file. This approach enables to properly verify the write-serializability.

It is worth highlighting that the malicious actions of the broker, and collusion attacks, aren't addressed in related work. During a collusion attack, the broker sends the attestation and metadata in accordance with an old file stored by the provider. Thus, it is possible, for example, to receive and accept an outdated file, because the attestation wrongly indicates that the file is up-to-date. On the other hand, sometimes, a user can detect the violation based on his/her last attestation. For example, a user requested a writing and stored the attestation of this transaction. Next, the user can detect a violation if he/she reads an old version of the file. However, this user can have his/her permissions revoked and does not perform new transactions. Thus, if the broker is compromised, the users may not detect, in monitoring, the violations because they do not have the correct information about the current state of the system, making the auditing indispensable to identify the undetected violations through historical analysis of cloud transactions.

Any involved entity can detect an external attack (e.g., replay attack) and cancel the unauthorized transaction, but a malicious behavior of the provider and/or broker results in a security violation. In this case, a penalty and a recovery procedure can be applied as specified in an SLA. For example, if an integrity violation is detected, it is possible to restore the most up-to-date version in a backup, reducing the damage. Therefore, the proposed mechanism monitor and audit security properties, detecting violations of integrity, freshness, and write-serializability.

5. Modeling and Validation

The use of formal methods is a good approach to model and validate the proposed mechanism, given the difficulty of demonstrating the security only with experiments in cloud infrastructures. We use the CPN Tools[2] to model this mechanism, because of

[2] http://cpntools.org

the ease of use of this tool and its extensive documentation, as well as the suitability of evaluating security protocols [Carvalho et al. 2016]. In this section, we present the modeling of the proposed, as well as, the scenarios to evaluate the robustness of this mechanism against existing threats and flaws found in related work. The modeled CPN includes the protocols for auditing and for reading and writing files in order to analyze the detection of violations.

A simplified CPN of this mechanism is available at https://sites.google.com/site/candrebc/scss.zip[3] and the overview of this modeling is shown in Figure 3. The simplifications do not change it functioning and include: i) the storage of a single file; ii) the execution of a single transaction each time; iii) the representation of the chain hash as a sequence number; and iv) the assumption that all messages are authentic. The modeled CPN is parametrized so that several users can be easily added making this mechanism theoretically scalable. Besides, the modeling indicates a violation when the system reaches one violate state, represented by the presence of tokens in error places of the CPN. These simplifications facilitate the analysis of the modeling due to the reduction of its complexity and of the space state.

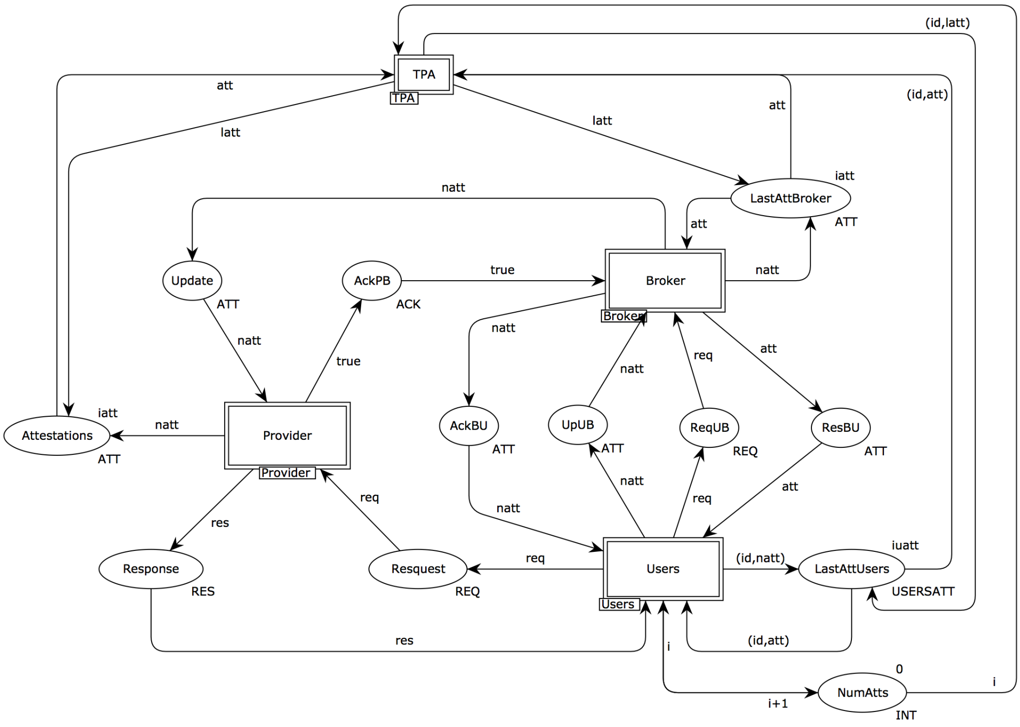

Figure 3. Overview of the simplified CPN

This modeling represents the correct behavior of the involved entities and simulations were realized to verify the correctness and to evaluate the modeled mechanism. Thus, none violation is detected because the analysis of the space state reveals that no token is stored in error places in any possible state. This result is expected when the entities are trustworthy, and this mechanism demonstrates the assurance of security properties of the modeled storage service.

[3] The original file can be used for proper analysis and simulations. A *.pdf* file is also available.

In order to validate the proposed mechanism, it is necessary to demonstrate that all possible attack is detected, especially that resulting from malicious actions by the provider and broker. These actions result in violations that must be detected by the users or the TPA. Then, we modeled malicious behaviors of the provider and broker in different CPNs, simulating each violation scenario. The specification of two users in each modeled CPN was enough to detect the violations. We also define a special place to indicate that the provider or broker execute malicious actions. Thus, every time that a violation is triggered, the mechanism should detect it.

Table 2 presents the results of the validation, reporting that the violations are always detected, in accordance with the attacks enumerated in Subsection 2.3. Although the users can identify previously the collusion attacks based on their last attestation that indicates the attestation sent by the broker is old, we inform in this table the worst case of violation detection.

Table 2. Violation detection

Provider	Broker	Detection time
Honest	Send an outdated file	Monitoring
Honest	Write out of order	Monitoring
Honest	Do not commit a writing	Monitoring
Send an old attestation	Send an outdated file	Auditing
Send an old attestation	Write out of order	Auditing
Send an old attestation	Do not commit a writing	Auditing
Send an old attestation	Honest	Monitoring

We modeled the rollback attack that restores the provider to a previous state of the system. It is possible to rollback only the files or the files and the attestations. In both cases, the sending of an outdated file is detected by the users when the broker is honest and informs the expected version of the file. However, if the broker colludes with the provider, sending an old attestation, the TPA will identify the violation.

If it is requested a writing after the rollback, the provider can only write a file out of order because the change of the file version results in an integrity violation. The absence of some file versions is detected by the TPA when ordering the attestations only if the provider restores to previous files and attestations. However, if the broker is honest, the write-serializability violation is detected in real-time when the verification of retrievability is included in all writing transactions. This verification fixes security flaws found in related work, being an important improvement of the violation detection mechanism. For simplicity, this step is represented in our modeling by the reading of the file, but it must be replaced by a PoR scheme as defined in our proposal.

In order to verify the retrievability, we modeled a provider that confirms the writing of the second version of a file, without committing it (i.e., the provider send the writing attestation, but keep only the first version of the file). It is visible that the violation is detected if the next transaction is a reading. Otherwise, if the next transaction is a writing, the first version will be overwritten by the third version, violating the write-serializability. Thus, no violation is identified even in the auditing because the attestations are rightly ordered in it. Then, it is necessary to verify what is the version of the file really stored, before requesting a writing. It is interesting to highlight that we analyzed the possibility of verifying the writing after performing it, but the data loss of a rollback attack could not be detected.

On the other hand, in a collusion attack, the broker informs to a user the same version of the file sent by the provider. In the worst case, the TPA reports the violation because there is a reduction of the file versions, or some UserLSN is absent. The detection of collusion attacks is not addressed by related work and is another improvement of our mechanism.

Loss of data, resulting from the malicious provider that do not commit a writing request, can trigger a freshness or write-serializability violation, depending on whether the next transaction is a reading or writing. In both cases, the violation is detected in a similar manner to the previous scenarios.

Besides, this mechanism detects malicious behaviors of the broker that can inform an old attestation. The users identify this behavior when the honest provider send a newer version of the requested file. The last attestation maintained by the provider is signed by also by the broker and can be used to prove the failure in the broker. The audit is only necessary to demonstrate if the provider also lost some information. If the broker loses the tags used by the PoR scheme, the file must be read to check its version.

External attackers can meddle in the communication to perform illegal transactions, according to Doley-Yao model [Dolev and Yao 1983]. The use of cryptography to sign the messages and the verification of the UserLSN make these attacks unfeasible. In addition, the access control mechanism is secure so that an unauthorized user cannot obtain the encryption and signatures keys. Thus, an unauthorized reading is not possible, and the provider can identify an unauthorized writing, based on the signature of the file. On the other hand, a malicious provider can commit this writing or own a corrupted file. Due to the absence of failures in integrity verification of the related work, we do not specify the execution of illegal transactions in Table 2. However, the resulting violations are detected during the monitoring since the security of cryptographic primitives is attested by the scientific community. In these cases, the integrity violation is detected when a user reads the corrupted file, or the challenge-response fails, during a writing.

We also observed how the modifications in the proposed mechanism affect the violation detection. For example, the users and the broker must keep their last attestations of all files even if a single chain of attestation is used for all files. The real-time detection is improved because the attestation about one file is not overwritten after a transaction with another file. If the UserLSNs is not verified in auditing, violations resulting from the loss of attestations in collusion attacks are not detected.

When analyzing the concurrent access, no violation occurs if simultaneous readings are requested. However, simultaneous writings can result in the commitments with the same FileVersion, or in the absence of some FileVersion if some transaction is not finalized. In addition, if a reading is requested before a writing ends, it is possible to receive the written version or the previous one.

Although the main goal is the detection of malicious acts of the provider, broker and external attackers, our modeling proves that some malicious actions of users are also identified. For example, when requesting a writing operation, a user can inform the wrong version of a file, and the broker and provider detect this act by knowing the expected version. However, it is indispensable the users rightly inform their last attestations to avoid that some violation is not identified by the auditing.

Lastly, it is important to highlight that the management of transaction by the broker enables the detection of violations in real-time. Our analysis demonstrates that violations are detected in real-time if the broker is honest. The audit is then necessary to solve any contestation and identify collusion attacks.

6. Conclusion

A secure cloud storage service must include a mechanism to monitor and audit security properties. With this mechanism, a provider proves the assurance of these properties, improving the transparency and trust of security storage services. During this research, we identify that existing mechanisms do not detect all security violations and fail in demonstrating the security. Besides, related work does not properly address the real-time detection and collusion attacks.

In this paper, we described a mechanism to verify integrity, freshness and write-serializability, fixing existing flaws. The modeling with CPNs enables a formal validation of the proposed mechanism and proves the need of auditing to detect collusion attacks. In the validation, the attacks were modeled and detected by this mechanism. As proposed in [Carvalho et al. 2017b], a secure cloud storage service must combine this mechanism with an access control mechanism to provide also the confidentiality, protecting the customers against data loss and data leakage while enabling the data sharing. The proposed mechanism can also be used with an SLA solution to provide these properties. However, it is necessary to define the penalties and contingency plan to reduce the damage when a violation is detected.

This paper focuses on improving the violation detection and on security evaluation. As future work, this mechanism could be deployed in a cloud infrastructure to evaluate its functioning in a real scenario and analyzing performance aspects. It is possible to analyze other access control mechanisms (*e.g.*, proxy re-encryption or Attribute-Based Encryption [Thilakanathan et al. 2014]), identifying what is most suitable for secure storage environment. A robust solution should also include mechanisms to address other security properties such as availability and location.

Acknowledgments

This work was partially supported by the STIC-AmSud project SLA4Cloud. Carlos André Batista de Carvalho was also supported by CAPES/FAPEPI Doctoral Scholarship.

References

Albeshri, A., Boyd, C., and Nieto, J. G. (2012). A security architecture for cloud storage combining proofs of retrievability and fairness. In *3rd International Conference on Cloud Computing, GRIDS and Virtualization*, pages 30–35.

Albeshri, A., Boyd, C., and Nieto, J. G. (2014). Enhanced geoproof: improved geographic assurance for data in the cloud. *International Journal of Information Security*, 13(2):191–198.

Bamiah, M. A., Brohi, S. N., Chuprat, S., and lail Ab Manan, J. (2014). Trusted cloud computing framework for healthcare sector. *Journal of Computer Science*, 10(2):240–240.

Carvalho, C. A. B., Andrade, R. M. C., Castro, M. F., and Agoulmine, N. (2016). Modelagem e detecção de falhas em soluções para armazenamento seguro em nuvens usando redes de petri coloridas: Um estudo de caso. In *XIV Workshop de Computação em Clouds e Aplicações (WCGA/SBRC)*, pages 17–30. in portuguese.

Carvalho, C. A. B., Andrade, R. M. C., Castro, M. F., Coutinho, E. F., and Agoulmine, N. (2017a). State of the art and challenges of security SLA for cloud computing. *Computers and Electrical Engineering*. In Press.

Carvalho, C. A. B., Castro, M. F., and Andrade, R. M. C. (2017b). Secure cloud storage service for detection of security violations. In *CCGrid'17 Doctoral Symposium*. Accepted for publication.

Celesti, A., Fazio, M., Villari, M., and Puliafito, A. (2016). Adding long-term availability, obfuscation, and encryption to multi-cloud storage systems. *Journal of Network and Computer Applications*, 59:208–218.

CSA (2013). The notorious nine: Cloud computing top threats in 2013. Technical report, Top Threats Working Group.

Dolev, D. and Yao, A. C. (1983). On the security of public key protocols. *Information Theory, IEEE Transactions on*, 29(2):198–208.

Hwang, G.-H., Huang, W.-S., and Peng, J.-Z. (2014a). Real-time proof of violation for cloud storage. In *CloudCom'14*, pages 394–399.

Hwang, G.-H., Huang, W.-S., Peng, J.-Z., and Lin, Y.-W. (2014b). Fulfilling mutual nonrepudiation for cloud storage. *Concurrency and Computation: Practice and Experience*.

Jin, H., Zhou, K., Jiang, H., Lei, D., Wei, R., and Li, C. (2016). Full integrity and freshness for cloud data. *Future Generation Computer Systems*.

Luna, J., Suri, N., Iorga, M., and Karmel, A. (2015). Leveraging the potential of cloud security service-level agreements through standards. *IEEE Cloud Computing Magazine*, 2(3):32 – 40.

Popa, R. A., Lorch, J. R., Molnar, D., Wang, H. J., and Zhuang, L. (2011). Enabling security in cloud storage slas with cloudproof. In *USENIXATC'11*.

Rong, C., Nguyen, S. T., and Jaatun, M. G. (2013). Beyond lightning: a survey on security challenges in cloud computing. *Computers and Electrical Engineering*, 39(1):47–54.

Thilakanathan, D., Chen, S., Nepal, S., and Calvo, R. A. (2014). Secure data sharing in the cloud. In *Security, Privacy and Trust in Cloud Systems*, pages 45–72. Springer.

Tiwari, D. and Gangadharan, G. (2015). A novel secure cloud storage architecture combining proof of retrievability and revocation. In *ICACCI'15*, pages 438–445.

Zou, H., Qian, Y., Zhao, Y., and Ding, K. (2015). The design and implementation of data security management and control platform. In *ATIS'15*, pages 368–378.