UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE CIÊNCIAS

DEPARTAMENTO DE COMPUTAÇÃO

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

CARLOS ANDRÉ BATISTA DE CARVALHO

SCUDO: SECURE CLOUD STORAGE SERVICE FOR DETECTING VIOLATIONS OF SECURITY PROPERTIES IN A DATA SHARING ENVIRONMENT

FORTALEZA

2018

CARLOS ANDRÉ BATISTA DE CARVALHO

SCUDO: SECURE CLOUD STORAGE SERVICE FOR DETECTING VIOLATIONS OF SECURITY PROPERTIES IN A DATA SHARING ENVIRONMENT

Tese apresentada ao Curso de Programa de Pós-Graduação em Ciência da Computação da Universidade Federal do Ceará, como requisito parcial à obtenção do título de doutor em Ciência da Computação. Área de Concentração: Sistemas de Informação

Orientadora: Profa. Dra. Rossana Maria de Castro Andrade

Co-Orientador: Prof. Dr. Miguel Franklin de Castro

FORTALEZA

2018

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

C322s Carvalho, Carlos André Batista de.
SCUDO: Secure CloUd storage service for Detecting viOlations of security properties in a data sharing environment / Carlos André Batista de Carvalho. – 2018.
109 f. : il. color.

Tese (doutorado) – Universidade Federal do Ceará, Centro de Ciências, Programa de Pós-Graduação em Ciência da Computação , Fortaleza, 2018.
Orientação: Profa. Dra. Rossana Maria de Castro Andrade.
Coorientação: Prof. Dr. Miguel Franklin de Castro.

1. 1. Secure storage. 2. Cloud security. 3. Monitoring and auditing. 4. Violation detection. I. Título.
CDD 005

CARLOS ANDRÉ BATISTA DE CARVALHO

SCUDO: SECURE CLOUD STORAGE SERVICE FOR DETECTING VIOLATIONS OF SECURITY PROPERTIES IN A DATA SHARING ENVIRONMENT

Tese apresentada ao Curso de Programa de Pós-Graduação em Ciência da Computação da Universidade Federal do Ceará, como requisito parcial à obtenção do título de doutor em Ciência da Computação. Área de Concentração: Sistemas de Informação

Aprovada em: 25 de Setembro de 2018

I dedicate this thesis to my wife Maíra and to my sons João Carlos e Pedro André.

ACKNOWLEDGEMENTS

First, I thank God and everyone who have encouraged and supported me.

Then, I would like to express my sincere gratitude to my advisor Prof. Rossana Andrade, for the competence and commitment in the development of this work. I also thank Prof. Miguel Franklin, for his co-orientation, and Prof. Nazim Agoulmine, who collaborated a lot for my learning.

Besides, I would like to thank my parents, Dimas and Isabel, for my education and support. Many thanks also go to my wife Maíra and my son João who accompanied me and motivated me during this journey. I thank also my youngest son Pedro and my brothers Dimas and Moara. I could not forget to thank my parents-in-law, friends and relatives, especially Fátima, Gildo, Domingos and Regina for the welcome in Fortaleza.

Almost last but not least, many thanks go to my friends from the GREat lab, with whom I overcame the obstacles of this course, and to you professors Danielo Gomes, Neuman Souza and Elias Junior for generously sharing your time and knowledge.

Finally, I also would like to thank UFPI, UFC, FCPC, FAPEPI and CAPES for the opportunity and financial support of this research. This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

“Não controla-se sonhos, controla-se o pensamento.”

(João Carlos C. de Carvalho)

ABSTRACT

A cloud storage service implements security mechanisms to protect user data. Due to the customer needs and existing threats, the secure data sharing is a key issue highlighted in the literature. Moreover, due to the loss of control over the cloud infrastructure, it is essential to design security mechanisms that focus on the trust and transparency of the cloud services. The confidentiality, integrity, freshness and write-serializability are the security properties analyzed in this research. Usually, auditing and monitoring mechanisms are used to detect violations of security properties. However, an analysis of the literature reveals attacks that are not identified by existing solutions. Although a broker has been used to enable a real-time detection, it is necessary to identify collusion attacks resulted from malicious actions of this broker. The detection of integrity violations has not been properly addressed, ignoring the violations that result from the writing transactions performed by revoked users. Similarly, the reading by revoked users implies in confidentiality violations that must also be detected. Last, the verification of write-serializability violations should be effective, identifying properly the violation's scenarios. Therefore, a secure storage service for cloud computing, called SCUDO, is proposed in this thesis to address these issues, improving the violation detection while allowing the data sharing. The detection of violations is based on the log of the performed transactions that is signed for purposes of non-repudiation. The evaluation of SCUDO is performed based on a formal model using Colored Petri Nets (CPNs) and a prototype deployed in a cloud infrastructure. The results show that the provider cannot deny a violation and attacks are detected as soon as possible, reducing the damage of an attack. Then, the security mechanisms at SCUDO can allow the provider and the broker to ensure security properties and show evidence that they are honest.

Keywords: Secure storage. Cloud security. Monitoring and auditing. Violation detection.

RESUMO

Um serviço de armazenamento na nuvem implementa mecanismos de segurança para proteger os dados dos usuários. Devido às necessidades dos clientes e às ameaças existentes, o compartilhamento seguro de dados é uma questão importante destacada na literatura. Além disso, devido à perda de controle sobre a infraestrutura de nuvem, é essencial projetar mecanismos de segurança focados transparência nos serviços em nuvem, aumentando a confiança nos mesmos. Normalmente, os mecanismos de auditoria e monitoramento são usados para detectar violações de propriedades de segurança. A confidencialidade, integridade, *freshness* e *write-serializability* são as propriedades de segurança analisadas nesta pesquisa. Uma análise da literatura revela ataques que não são detectados pelas soluções existentes. Um broker pode ser utilizado para viabilizar a detecção de violações em tempo real. Contudo, é necessário identificar ataques de conluio (*collusion attacks*) resultantes de ações maliciosas desse broker. Além disso, a detecção de violações de integridade não tem sido tratada adequadamente, ignorando a possibilidade de usuários cujas permissões foram revogadas escreva arquivos usando chaves antigas. Similarmente, é possível que usuários revogados consigam ler arquivos, violando a confidencialidade dos dados. Por fim, a verificação de *write-serializability* deve identificar adequadamente os cenários de violação existentes. Neste contexto, este trabalho propõe um serviço de armazenamento seguro para computação em nuvem, denominado SCUDO, melhorando a detecção de violações enquanto permite o compartilhamento de dados. Esta detecção de violações é baseada no log das transações realizadas que é assinado para prover o não repúdio dessas transações. A avaliação do SCUDO é feita com base em uma modelagem formal utilizando Redes de Petri Coloridas (CPNs), que é essencial para avaliar a segurança da solução proposta, e em um protótipo implantado em uma infraestrutura de nuvem. Como resultado, o provedor não pode negar uma violação e os ataques são detectados o mais rápido possível, reduzindo o dano desses ataques. Então, os mecanismos de segurança existentes no SCUDO permitem que o provedor e o *broker* ofereçam garantias quanto às propriedades de segurança e mostrem evidências que são honestos.

Keywords: Armazenamento seguro. Monitoramento e auditoria. Detecção de violações.

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

ABE	Attribute-Based Encryption
ACL	Access Control List
AES	Advanced Encryption Standard
API	Application Programming Interfaces
AWS	Amazon Web Services
CPN	Colored Petri Net
CPN ML	CPN Markup Language
CSA	Cloud Security Alliance
CTL	Computational Tree Logic
ISO	International Organization for Standardization
KMS	Key Management Service
LOTOS	Language Of Temporal Ordering Specification
LSN	Last Sequence Number
LTL	Linear Temporal Logic
LTS	Long Term Support
MAC	Message Authentication Code
NIST	National Institute of Standards and Technology
NS	Needham-Schroeder
PoR	Proof of Retrievability
PRE	Proxy Re-Encryption
QoS	Quality of Service
RSA	Rivest-Shamir-Adleman
SCUDO	Secure CloUd storage service for Detecting viOlations of security properties
SDL	Specification and Description Language
SecSLA	Security Service Level Agreements
SLA	Service Level Agreement
SSL	Secure Sockets Layer
TLS	Transport Layer Security (TLS)
TPA	Third-Party Auditor
TTP	Trusted Third-Party
UML	Unified Modeling Language

XML	Extensible Markup Language

CONTENTS

1 INTRODUCTION

This thesis presents a secure cloud storage service that verifies security properties, detecting attacks related to data confidentiality, integrity, freshness and write-serializability. This research improves the violation detection while allowing data sharing, based on the analysis of attacks that are not identified by existing solutions.

In this chapter, the context of this research is introduced, highlighting the existing gaps that are addressed in this thesis. Then, the hypothesis and the research questions are presented as well as the objectives and the methodology used during the development of this work. At the end of the chapter, the structure of this document is detailed.

1.1 Research context

Cloud computing is a distributed computing paradigm that enables the sharing of computational resources among many clients (BUYYA *et al.*, 2009). It is possible to reduce the infrastructure costs by contracting a public cloud provider and paying only for the consumed resources. Besides, scalability and elasticity allow the dynamic allocation of resources, in accordance with customers' needs.

Cloud computing is an object of investment of big companies, and diverse cloud services have already been deployed. However, this paradigm comes with the main drawback of losing control over the cloud infrastructure so that a provider may act maliciously and even deny the occurrence of an attack (SARIPALLI; WALTERS, 2010). Consequently, there is resistance, in society, to adopt public clouds, due to concerns about security and privacy (LUNA *et al.*, 2015). Sun *et al.* (2014) highlight several security challenges related to, for example, data loss, data segregation, data sharing, data location, service disruption, malicious attacks and multi-tenancy issues.

Cloud providers develop 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). For example, Amazon S3 (Simple Storage Service) uses the SSL (Secure Socket Layer) protocol to protect data transmission and the AWS KMS (Key Management Service) to protect data at rest (AMAZON, 2018a). KMS facilitates the creation, storage and distribution of keys. In addition, the permission management, based on Access

Control Lists (ACLs), enables secure data sharing (AMAZON, 2018b).

However, customers have a limited view of security and can request mechanisms to provide service security guarantees (LUNA *et al.*, 2015). One possibility is to define a legal contract, called Service Level Agreement (SLA), in which the cloud provider specifies its level of Quality of Service (QoS) assurance through the parameters of the non-functional requirements (*e.g.*, availability and performance) (BOSE *et al.*, 2011). However, security terms have not been covered by SLAs of public cloud providers that frequently specify only the service availability such as the SLA of Amazon S3[1].

Therefore, it is necessary to develop solutions to assure security properties, enabling the definition of Security Service Level Agreements (SecSLA) (HABIB *et al.*, 2011). Thus, it is possible to improve the trust and transparency of cloud providers. In addition, a SecSLA must be attested by monitoring and auditing mechanisms to validate the security provided or to show the occurrence of any violation.

Rong *et al.* (2013) stress the assurance of security properties in cloud storage as a relevant concern. Usually, confidentiality, integrity and availability are the required security properties, but the literature highlights other properties related to secure cloud storage, such as freshness (POPA *et al.*, 2011), write-serializability (POPA *et al.*, 2011), retrievability (TIWARI; GANGADHARAN, 2015a) and location (ALBESHRI *et al.*, 2014).

Security mechanisms are designed to prevent or detect attacks, and the success of an attack results in the violation of some security property (STALLINGS, 2016). For example, an access control mechanism is used, in a data sharing environment, to allow the definition and updating of files' permissions. Usually, this mechanism includes cryptographic solutions to provide confidentiality, preventing data leakage. Cryptography is essential to avoid unauthorized access to files by the providers (TIWARI; GANGADHARAN, 2015b). In this context, the key management becomes essential, being responsible for updating the keys and their distribution only for authorized users (POPA *et al.*, 2011).

Nevertheless, some attacks, performed by malicious insiders (*e.g.*, users and providers), cannot be avoided, resulting in security violations. A malicious provider can execute illegal transactions and, for example, write files of unauthorized users or send old versions of files. Therefore, cloud customers are requesting more transparency and security guarantees from providers (ARDAGNA *et al.*, 2015). In this context, research has been done to develop auditing

[1] http://aws.amazon.com/s3/sla/

and monitoring solutions that improve the trust in cloud providers (JIN *et al.*, 2018) (HWANG *et al.*, 2014) (TIWARI; GANGADHARAN, 2015a). These solutions can detect malicious behaviors that result in violations of security properties.

Existing solutions are usually based on auditing mechanisms to demonstrate the security and allow the violation detection (POPA *et al.*, 2011) (WANG *et al.*, 2013). On the other hand, it is more useful to monitor the security properties, because it is possible to anticipate the detection of a violation or even allow blocking an attack, reducing the damage (HWANG *et al.*, 2014). Meanwhile, the auditing is essential to solve any dispute (*i.e.*, to prove false accusations) or identify violations not detected by monitoring (ALBESHRI *et al.*, 2012b).

In a secure cloud storage service, security mechanisms are combined to treat different security issues in a complete solution. For example, Popa *et al.* (2011) describe a secure storage service based on Access Control Lists (ACLs), called CloudProof, in which integrity violations are detected while reading files if verification of their signatures fails. Cloud transactions are also audited to verify freshness and write-serializability. Hwang *et al.* (2014) specify a Trusted Third-Party (TTP), called synchronization server, to enable real-time violation detection.

1.2 Motivation

Solutions for access control have been focused on providing confidentiality and assume that the provider is "honest but curious" (TIWARI; GANGADHARAN, 2015b) (JIANG *et al.*, 2014). Thus, the provider rightly follows the system specification but can try to access sensitive information (XIA *et al.*, 2016). On the other hand, a provider can be malicious and perform an illegal writing transaction. This malicious action can be detected when a corrupt file is read, validating its signature based on a writing key. This key is updated by the cloud customer when changing the permissions of a file in order to avoid writing by revoked users. However, a revoked user can try to write a new version of this file using an old key (KALLAHALLA *et al.*, 2003). Other users may not be aware of the modification of the permissions, not identifying the integrity violation if the malicious provider ignores the access control and commits this writing.

While data integrity is checked during the reading of files, the freshness and write-serializability are verified during the auditing (HWANG *et al.*, 2016) or in real-time (JIN *et al.*, 2018). Real-time solutions monitor security properties, assuming the use of a TTP or an honest broker that manages the cloud transactions (*e.g.*, reading and writing). This third-party is responsible to inform users about the current state of the system, enabling the real-time detection of violations. However, due to this assumption, a new threat is the possibility that the third-party be also malicious, colluding with the provider to execute attacks and violate the security properties.

Another security problem results from the lazy revocation approach, which is used to improve performance when updating permissions (KALLAHALLA *et al.*, 2003). In this approach, the files are re-encrypted with new keys only in subsequent writing. Therefore, a malicious provider can send a file before its update, and a revoked user has the key to access it. The detection of this data leakage is not analyzed in the literature.

In this context, it is essential to identify the attacks that violate the aforementioned security properties and to analyze how the violations can be detected. This analysis can be used to design an improved secure storage service, in order to fix identified flaws and detect security violations in real-time, whenever is possible. However, collusion attacks can be identified only in auditing due to the conspiracy between the involved entities, which try to deceive the violation detection mechanism. The existence of attacks that cannot be identified during monitoring reinforces the importance of the auditing. The auditing is also used to prove the violations so that a provider cannot deny a detected violation.

In short, the literature review, done in this work and summarized in this section, highlighted the limitations and flaws of existing solutions, which are rarely deployed in real cloud infrastructures and formally evaluated. In a preliminary study of this thesis (CARVALHO *et al.*, 2016a), CloudProof (POPA *et al.*, 2011), mentioned in the previous section, was modeled using Colored Petri Nets (CPNs) and this model shows the existence of non-detected write-serializability violations. These undetected violations result from the loss of a file followed by writing a new version of this file. Other solutions, which appear in the literature, are also limited and show that there is a need for the development of a secure storage service.

1.3 Hypothesis and research questions

Due to the limitations of existing solutions for data storage and sharing in cloud environments (see Section 1.2), this thesis investigates the following **research hypothesis**: *it is possible to design and combine security mechanisms in a secure cloud storage service to improve the violation detection of confidentiality, integrity, freshness and write-serializability in a data sharing environment.*

Based on this hypothesis, the following Research Questions (RQ) are extracted to be investigated during this work:

- **RQ01**: What are the requirements of secure cloud storage service to allow data sharing and the violation detection of confidentiality, integrity, freshness and write-serializability security properties?
- **RQ02**: What are the attacks that violate these properties and which ones are not detected by existing solutions?
- **RQ03**: How can a storage service be monitored and audited to verify the security properties?
- **RQ04**: Are there attacks in which the resulted violations cannot be identified in real-time, requesting an auditing to detect these violations?
- **RQ05**: How can security mechanisms be evaluated to demonstrate the security of the storage service, ensuring the non-violation of security properties or proving the occurrence of an attack?

1.4 Research goal and main contributions

The main goal of this thesis is, thus, *to propose a secure storage service in which security properties of the data stored in the cloud are ensured while allowing file sharing*. In order to provide security guarantees, an access control mechanism must be combined with monitoring and auditing mechanisms to detect violations of these properties resulting, especially, from malicious actions on the provider.

The focus of this thesis is on the design of mechanisms to verify security properties, enabling them to show if some property is violated or not by a cloud storage service. To this end, Secure CloUd storage service for Detecting viOlations of security properties (SCUDO) is proposed, combining security mechanisms to ensure security properties and to allow the data sharing. To this end, the following targets are defined:

- Identification of the requirements of a secure cloud storage service, including the existing threats and attacks (related to RQ01);
- Analysis of the related work, highlighting the proposed approaches, limitations, and aspects related to the deployment and evaluation (related to RQ01 and RQ02);
- Proposal of a secure storage service, composed of mechanisms to verify security properties that improve the violation detection, identifying attacks not addressed in related work (related to RQ03 and RQ04));

- Formal specification of the proposed service and its validation (related to RQ05);
- Development and evaluation of a prototype deployed in the private cloud of the Group of Computer Networks, Software Engineering and Systems (GREat)[2] at the Federal University of Ceará (UFC) (related to RQ05).

This thesis does not aim to address all security aspects and focuses on mechanisms to ensure the confidentiality, integrity, freshness and write-serializability of the data stored in the cloud as well as other violation detection aspects as shown in Figure 1.

Although the verification of security properties is based on log analysis as described in other studies (POPA *et al.*, 2011) (TIWARI; GANGADHARAN, 2015a), new violation detection scenarios are addressed in the proposed storage service as shown in Figure 1. This figure highlights the expected contributions of this research, showing where the mechanisms that need to be developed (in black) or refined (in grey) in order to properly detect the violations of the security properties.

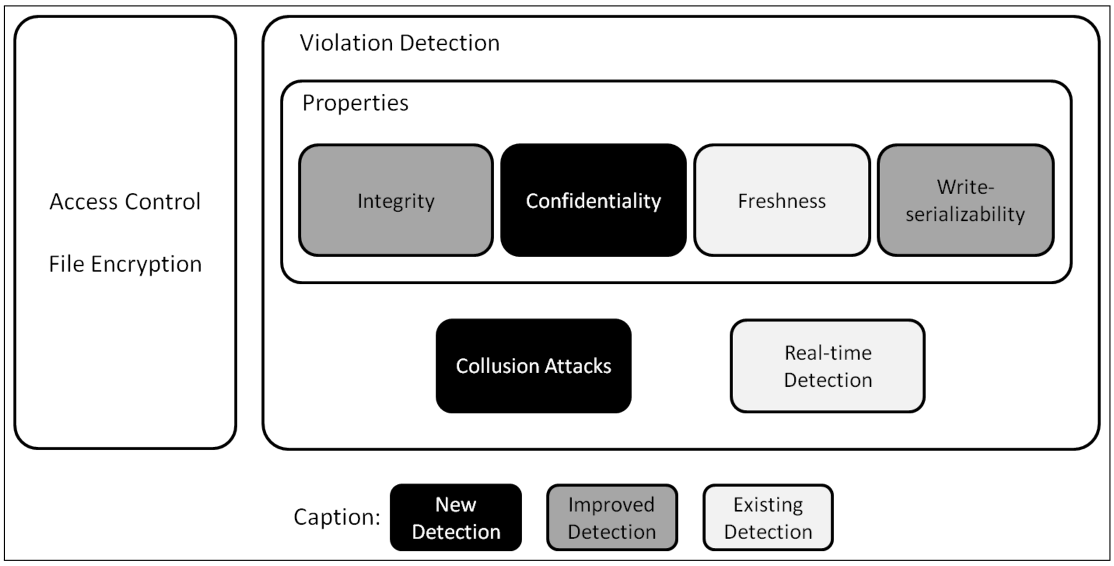
An analysis of the literature shows flaws in detecting violations related to integrity and write-serializability (both in grey in Figure 1), and the absence of mechanisms for detect confidentiality violations (in black). The detection of integrity violations (in grey) has not been properly addressed, ignoring the violations resulting from the writing transactions performed by revoked users (KALLAHALLA *et al.*, 2003). Similarly, the reading by revoked users implies in confidentiality violations (in black) that must also be detected. Last, the verification of write-serializability (in grey) should depend not only on the transaction logs but also on the verification of the stored version of a file before its updating.

In addition to detect violations of these properties, it is necessary to observe the capacity of a security mechanism to identify the violations in real-time (in light grey in Figure 1) and the possibility of the occurrence of collusion attacks (in black). Although a synchronization server (in this thesis called broker) has been used to enable real-time detection (HWANG *et al.*, 2014), it is necessary to identify collusion attacks (in black) resulting from malicious actions on this server. These attacks have not been addressed in literature.

Figure 1 also shows access control and file encryption that have mechanisms which are essential to enable data sharing while protecting against unauthorized access. For example, the use of Access Control Lists (ACLs) are appropriate to specify the users' permissions in solutions related to violation detection in a cloud storage environment (POPA *et al.*, 2011). In

[2] http://www.great.ufc.br

Figure 1 – Security overview of SCUDO



Source – The author.

this thesis, existing access control and file encryption mechanisms are used.

As a result of this thesis, the detection of violations is improved using the proposed storage service by addressing attacks scenarios that are not identified by related work. Thus, the proposed mechanisms can be used by an honest provider to offer security guarantees to its customers. These solutions to verify security properties, showing to cloud clients that the provided service is trustworthy, are important due to the loss of control of the customers using cloud infrastructures.

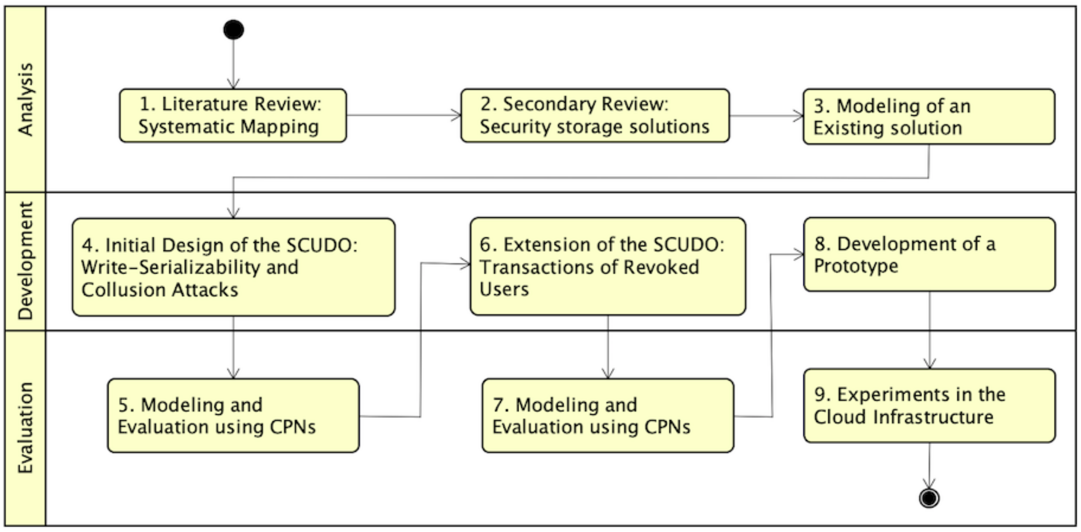
1.5 Research methodology

The research methodology used in this thesis is presented in Figure 2 that describes the activities performed to achieve the targets detailed in Section 1.4.

The *Analysis* phase contains the activities related to the definition of the security properties assurance in cloud storage, which is the subject of this thesis. First, a systematic mapping was done to bring an overview of existing solutions and open research opportunities in SecSLA for cloud computing. Numerous initiatives offer secure storage, but only a small number of them aims to provide security guarantees through an SLA.

In the second activity, a detailed analysis of related work, based on a non-systematic review, was done to understand the existing mechanisms and to identify the requirements and threats in the secure cloud storage scenario. This analysis also revealed the limitations of existing

Figure 2 – Research methodology


Analysis
Development
Evaluation
1. Literature Review: Systematic Mapping
2. Secondary Review: Security storage solutions
3. Modeling of an Existing solution
4. Initial Design of the SCUDO: Write-Serializability and Collusion Attacks
6. Extension of the SCUDO: Transactions of Revoked Users
8. Development of a Prototype
5. Modeling and Evaluation using CPNs
7. Modeling and Evaluation using CPNs
9. Experiments in the Cloud Infrastructure

Source – The author.

solutions and shows evidences of the existence of security flaws in the verification of the write-serializability property. The next activity was the formal modeling of an existing solution, the CloudProof (POPA *et al.*, 2011), which demonstrated its security flaws (CARVALHO *et al.*, 2016a). In this activity, CPN (JENSEN; KRISTENSEN, 2009) is chosen as the formal method to evaluate the security of SCUDO, because of the ease of use of the CPN Tools[3], the extensive documentation on CPNs, and its suitability for specifying security protocols (SEIFI, 2014).

After the *Analysis*, the *Development* and the *Evaluation* phases are executed using an incremental approach. First, the security mechanisms used by SCUDO are defined in the initial design. These mechanisms are related mainly to the verification of security properties that is based on the monitoring of cloud transactions and auditing. The modeling and validation of this design highlight its capacity to detect collusion attacks and properly identify write-serializability violations.

In the next step, this solution is extended to address the detection of violations resulting from transactions initiated by revoked users. After the modeling of the extended version, a prototype is developed to enable the execution of experiments in a cloud infrastructure. These experiments highlight the performance of the monitoring and auditing mechanisms as well as the capabilities of the system to detect violations in the attack scenarios.

[3] http://cpntools.org

1.6 Structure of the thesis

This thesis is structured in six chapters, starting with the introduction presented in this chapter. The remaining chapters are summarized as follows.

Chapter 2 exposes the background related to secure cloud storage, detailing the security properties addressed in this research and the attacks that result in their violations. This chapter also includes concepts about CPNs, the formal method used in this research to evaluate the security of the proposed mechanisms.

In Chapter 3, the related work is described, and the limitations of existing strategies are identified and discussed.

Next, Chapter 4 describes the thesis proposal, which is a secure cloud storage service, and specifies the used security mechanisms.

Chapter 5 details the evaluation of the proposed solution based on the formal modeling and the developed prototype.

Finally, Chapter 6 is dedicated to the final remarks of this thesis, highlighting the contributions and publications as well as future research directions.

2 BACKGROUND

Security is an important aspect to be addressed in cloud environments. Regarding secure storage, the main concerns are related to data corruption, loss and leakage (CSA, 2013). In this chapter, some of the major security issues of cloud storage are described in Section 2.1, including the security properties detailed in Section 2.2, and the existing threats addressed in this thesis (see Section 2.3). Moreover, a brief description of Colored Petri Net (CPN), a formal method used in this work, is presented in Section 2.4, exemplifying its use to evaluate security protocols formally.

2.1 Secure cloud storage

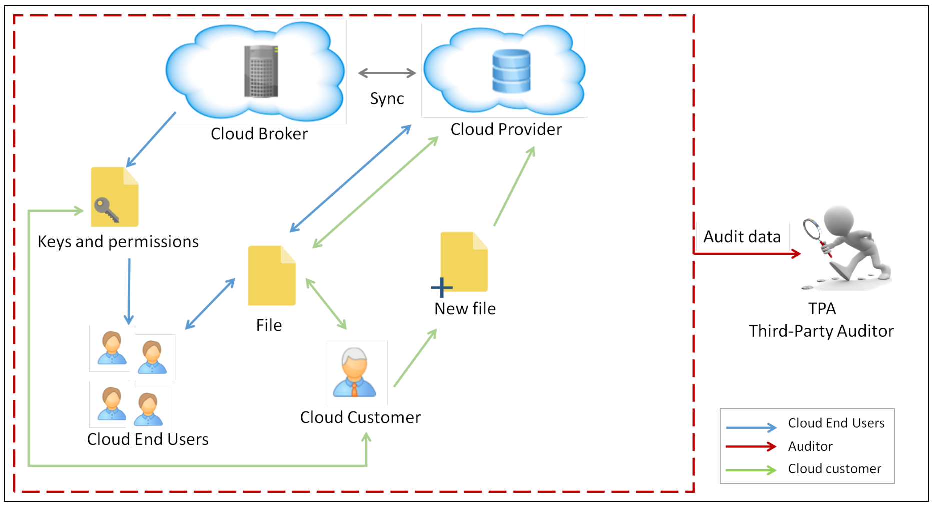
Cloud computing is a distributed computing paradigm that enables the sharing of computational resources among many clients (BUYYA *et al.*, 2009). Data storage is an example of a resource offered by cloud providers. A secure cloud storage service implements mechanisms for protecting the stored data from security attacks.

Security mechanisms should be designed to protect user data, avoiding attacks. However, the attacks cannot always be prevented, especially those from the malicious actions of users and providers. Due to the lack of control of public cloud infrastructures, it is essential to offer mechanisms to verify security properties of a cloud storage solution, providing security guarantees as well as the detection of security violations (TRAPERO *et al.*, 2017).

The essential features of a cloud storage service are the writing and the reading of files. A writing can be performed to upload a new or modified, while the file content is downloaded in a reading. Popa *et al.* (2011) suggested that the removal of files can be done when writing an empty file.

The definition of who can read or write each file, which is essential in a data sharing environment, is achieved by access control mechanisms. The control of concurrent transactions is also an important issue addressed by storage services (HWANG *et al.*, 2016). The file/folder management, keyword searching and version control are examples of other cited features of a storage service (CHANG *et al.*, 2016) (LIU *et al.*, 2012). The security mechanisms must be implemented to protect stored data and, in the scope of this research, enable secure sharing and verification of security properties. In order to satisfy these requirements, security mechanisms are combined, offering security, transparency and trust in cloud storage services. Thus, a secure

Figure 3 – Storage service model



Source – The author.

storage service has to include an access control mechanism, allowing the definition of permissions for each file, and monitoring/auditing mechanisms to verify security properties, enabling the detection of violations.

Figure 3 presents an overview and the stakeholders of a cloud storage service, and Table 1 describes the roles of each actor of this service. A cloud customer, also known as the data owner, can purchase this service, uploading new files and defining permissions for cloud end users that can, at least, read and modify them. The cloud transactions are managed by a broker and executed by a cloud provider. Due to the possibility that the provider and/or the broker act maliciously, the security properties are monitored by the users and audited by a Third-Party Auditor (TPA).

Some existing solutions do not include a broker explicitly as a stakeholder, and the provider is therefore responsible for the management of transactions (POPA *et al.*, 2011) (HWANG *et al.*, 2016). However, it is crucial to monitor security properties in real-time (HWANG *et al.*, 2014) and allows the storage of files in a multi-provider environment.

An access control mechanism aims to protect the storage service against unauthorized reading and data modification. Moreover, the cloud customer can eventually update the files' permissions, granting or revoking privileges to users. Usually, cryptographic solutions are used to provide confidentiality, preventing access to sensible information by providers and brokers (JIN *et al.*, 2018) (THILAKANATHAN *et al.*, 2014).

In addition to access control mechanisms, existing solutions include monitoring or auditing mechanisms to verify security properties (STAMOU *et al.*, 2013). In monitoring, a

Table 1 – Cloud actors definition

Actor	Definition
Cloud Customer	The person of an organization that administers the storage service. He/she is responsible for specifying the permissions and uploading new files. The customer is a user with administrative rights.
Cloud End User	An employee that can read or write files in accordance with his/her permissions. An end user also checks the security properties based on the information received by the broker
Cloud Broker	The entity responsible for managing the transactions. It checks the permissions before sending the keys and the current status of a file.
Cloud Provider	The entity responsible for storing the files.
Third-Party Auditor (TPA)	The entity responsible for analyzing the transactions' logs to demonstrate the security or prove the occurrence of a violation.

user analyzes logs and the messages exchanged during a cloud transaction to verify security properties. Also, the involved entities must verify the authenticity of the messages sent through the service. Thus, it is also possible to detect external attacks. The TPA performs audits when requested by the customer, identifying violations not detected in the monitoring. Moreover, an auditing can be executed when a violation is reported by a user to identify whether this user falsely accused the cloud provider.

An auditing 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. Usually, a TPA uses public keys to perform the verification without breaking the privacy (WANG *et al.*, 2013) (WORKU *et al.*, 2014). It is also possible to protect user identity by using anonymization techniques (MA; ZHANG, 2015). The auditing is essential to detect violations in the absence of an honest third-party.

Efficiency is a common concern of security mechanisms due to the overhead added by them. So, it is essential to analyze the extra cost of computation, storage and communication added by security mechanisms (TIWARI; GANGADHARAN, 2015b) (LI *et al.*, 2013). The efficiency of a solution cannot be significantly degraded in accordance with the amount of data and users, resulting in a scalable solution (LI *et al.*, 2013) (POPA *et al.*, 2011). In this context, the cost of auditing procedures can, for example, be reduced by batch auditing, in which multiple auditing requests are handled simultaneously (WANG *et al.*, 2013).

Brokers and cloud providers 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 also improves service availability (CELESTI *et al.*, 2016).

Besides the issues aforementioned, the literature describes other security aspects related to cloud storage. For example, Modi *et al.* (2013) also cite the following aspects: data segregation, data privacy, data recovery, data location and investigative support. From these aspects, it is interesting to indicate the use of trusting computing in assisting investigative support. Solutions of trusting computing are used by accountability tools to register all events in logs that are tampered-free and cannot be deleted (KO *et al.*, 2011).

2.2 Security Properties

Security properties represent the requirements that must be achieved by security mechanisms (LOPEZ *et al.*, 2009). In this thesis, the confidentiality, integrity, freshness and write-serializability of the data stored in the cloud are analyzed, dealing with data leakage, corruption and loss. Availability is another commonly required property, but it is out of the scope of this research.

Confidentiality and integrity are essential to avoid data access or data modification by unauthorized users. Freshness indicates that any reading is from the most recently updated version of any file. Write-serializability controls the writing order, ensuring that the new version of a file overwrites the last version of it.

Confidentiality is addressed through a mechanism of access control, based on cryptographic protocols, to avoid data leakage, including by storage providers, brokers, auditors and external attackers. A key management scheme is used to distribute the keys only to authorized users so that providers do not have access to plaintext.

Access control is also responsible for verifying the writing permissions so that unauthorized users cannot modify a file. Usually, authorized users sign the updated file, enabling integrity verification, based on digital signatures or Message Authentication Codes (MACs). These cryptographic primitives are attested by the scientific community so that the data corruption is identified by any involved entity. Thus, when a malicious provider commits an unauthorized writing transaction, the users can detect this malicious behavior while reading a corrupted file.

The verification of freshness and write-serializability is inherent in the operations of, respectively, reading and writing. Usually, this verification is performed based on logs that are signed by the involved entities, providing non-repudiation of the cloud transactions (HWANG *et al.*, 2016) (FENG *et al.*, 2011) so that the provider cannot deny a detected violation. After a data loss, the provider can, for example, restore the old files from a backup, without reporting this loss to users.

2.3 Threats and Attacks

A security threat indicates the possibility of the occurrence of attacks that violate some security property (STALLINGS, 2016). For example, data leakage is the threat related to the attack of traffic capture that affects the confidentiality of messages exchanged on a computer network. Security mechanisms are designed to avoid, detect or recover from an attack.

In this thesis, the focus is on monitoring and auditing mechanisms that increase the trust and transparency of cloud storage services by enabling the detection of attacks or demonstrating the security of the service. First, it is necessary to identify the attacks which can violate security properties, and then test their detection by these mechanisms. If some attack is not identified during the monitoring, the auditing must detect it. The attacks result from malicious acts of the external attackers and stakeholders, including end users.

In accordance with the Dolev-Yao model (DOLEV; YAO, 1983), an external adversary can impersonate any entity (user, broker or provider) by creating or modifying messages. However, a corruption attack is easily identified by integrity verification. Due to encryption, an attacker cannot understand the content of a message. Nevertheless, he/she can store this message and send it during a replay attack in order to, for example, overwrite a file with its previous version. Thus, the messages usually include timestamps or sequence numbers to allow the identification of old messages.

In this context, the stakeholders include verifications to prevent data corruption and replay attacks. On the other hand, a malicious provider can ignore these verifications and execute an illegal transaction. This behavior enables the successful execution of attacks, making it necessary to offer tools to monitor and audit the storage service, indicating the security of this service. Thus, it can be proved that the provider honesty or the security violations. Based on Hwang *et al.* (2016), Yang and Jia (2012), and Carvalho *et al.* (2016a), Table 2 highlights the malicious actions that can be performed by the provider, also showing the related security threat and property.

Table 2 – Malicious behaviors of the provider

Malicious Action	**Threat**	**Security Property**
Send an outdated file	Data loss	Freshness
Write files out of order	Data loss	Write-serializability
Perform an unauthorized writing	Data corruption	Integrity
Perform an unauthorized reading	Data leakage	Confidentiality

Sending outdated data and writing files out of order can occur after data losses, when, for example, the provider rollbacks the system to a previous state, using an old backup (HWANG *et al.*, 2016). Therefore, a user must check the data freshness or write-serializability when reading or writing a file. Moreover, successful replay attacks can also result in freshness or write-serializability violations. The detection of these violations depends on the management of the file versions. The broker is required to inform the current state of the file and to enable real-time monitoring in a data sharing environment.

The requests for unauthorized transactions come from external attacks or unauthorized users. An access control mechanism provides the credentials to read or write files only to authorized users, addressing the data confidentiality and integrity respectively. Although confidentiality is ensured by encryption, the files are not immediately re-encrypted if the lazy revocation approach is applied. Since it is not possible to prevent a malicious provider from sending files to unauthorized users, revoked users can use old keys to access files before they are updated (KALLAHALLA *et al.*, 2003).

Unauthorized writing results in a corrupted file so that a user can detect the attack, in the next transaction, when verifying the file signature. However, a malicious provider can try to deceive the violation detection if illegal writing is requested by a revoked user (KALLAHALLA *et al.*, 2003). In this scenario, this user uses an old writing key to sign a file, and the provider commits this writing. Next, this file can be considered authentic by other users if they do not know about updating the writing key. In this context, the execution of unauthorized transactions occurs when the provider ignores the access control, allowing a revoked user to read or write files. Thus, it is necessary to verify if each transaction is authorized.

It is also important to analyze the possibility of a broker being malicious. In this case, the malicious behavior occurs when the broker informs a previous state of the storage service. If only the broker is malicious, this behavior is identified by inconsistency when comparing to data sent by the honest provider. However, in a collusion attack, at least two entities act maliciously to execute an attack (TIWARI; GANGADHARAN, 2015b) (MA; ZHANG, 2015). If the provider and the broker are malicious, the users can not detect the violation. For example, during a rollback attack, the provider restores the system to a previous state, and the broker informs this old state to deceive the violation detection mechanism.

Besides, the broker and provider can also collude with an unauthorized user to provide access without permission. Although an access control mechanism ensures that an unauthorized user cannot obtain the current credentials, a revoked user, in collusion with the broker and provider, can write files using old credentials. Other users do not detect the violations because the dishonest broker informs that the stored file is correct. In this context, auditing is essential to identify the collusion attacks.

In addition, auditing allows for solving any conflict due to the non-repudiation of cloud transactions. The provider and broker can be, for example, in different states, and the auditing will report which entity is in the wrong state.

The previous example reveals the existence of malicious acts from the end users. The non-identification of these acts by the provider or broker results in security violations. Besides trying to write files without permission, a user can request a writing, informing a wrong version of a file. The provider can detect this action by knowing the expected version. It is essential to prove, in the security evaluation, the protection against the enumerated threats.

2.4 Security validation

Formal methods can be used to specify and validate systems, showing the absence of errors in these systems and compliance with their requirements (CLARKE; WING, 1996). Petri Nets, Estelle, LOTOS (Language Of Temporal Ordering Specification) and SDL (Specification and Description Language) are examples of formal languages used to specify systems (ARDIS, 1997). However, the evaluation of existing solutions for secure cloud storage is generally limited to informal discussions about their security and performance analysis (POPA *et al.*, 2011) (HWANG *et al.*, 2016) (JIN *et al.*, 2018). Thus, some security flaws may not be identified due to the absence of formalisms in the evaluation as detailed in Carvalho *et al.* (2016a).

In this context, Armando *et al.* (2014) describe studies in which formal methods are used to demonstrate the security or prove flaws of protocols, Application Programming Interfacess (APIs) and business processes. For example, Pommereau (2010) and Seifi (2014) detail flaws in the Needham-Schroeder (NS) protocol using CPNs (Colored Petri Nets). NS is a protocol to provide mutual authentication using symmetric or asymmetric cryptography (NEEDHAM; SCHROEDER, 1978). The next two subsections, respectively, introduce CPNs and their use to model and validate security protocols using, as an example, the version based on asymmetric cryptography of this protocol (POMMEREAU, 2010; SEIFI, 2014).

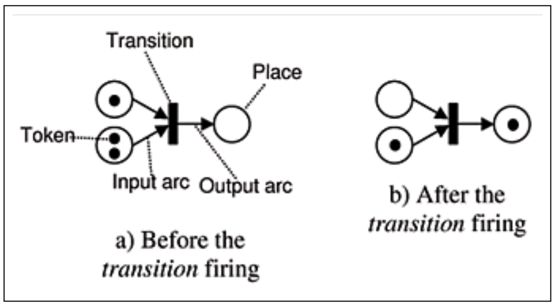
2.4.1 Colored Petri Nets

CPN is a general purpose formal language, with syntax and semantics mathematically defined, making possible the elimination of existing inconsistencies in the system specification. **CPN Tools**[1] is widely used to model systems graphically with the help of a programming language, called **CPN ML (Markup Language)** (JENSEN; KRISTENSEN, 2009). The graphic representation of a model makes it easier to understand.

In this research, CPN Tools have been used to formally model and validate the proposed mechanisms, because of the ease of use of this tool, of the extensive documentation on CPNs, and its suitability for specifying security protocols. Seifi (2014) highlight the limitations of some tools for analyzing security properties. Besides, the preliminary study proves the feasibility of using this tool (CARVALHO *et al.*, 2016a), with the modeling and validation of an existing solution.

The modeling of a system in Petri Nets is done by describing a graph composed of nodes that indicate the places and the transitions of the system (see Figure 4). Arcs connect places to transitions or vice versa, but never two places or two transitions. The places in a Petri Net contain tokens, which can enable transitions to be fired. When a transition occurs, tokens from the input places are consumed, and new tokens are added at the output places, according to the arc expressions. The choice for the transition to be fired is non-deterministic. The initial position of the tokens in places is called the initial marking, and each marking during the execution of a system represents a state of this system.

Figure 4 – Example of Petri Net


Transition
Place
Token
Input arc
Output arc
a) Before the *transition* firing
b) After the *transition* firing

Source – (VILLANI; MIYAGI, 2004).

The Colored Petri Net is an extension of the Petri Net, which allows assigning colors or values to the tokens. Similar to programming languages, it is possible to define what type of

[1] http://cpntools.org

information can be stored in each place and also assign or verify the values of the tokens. Thus, the size of the created models is reduced. There are other extensions, including hierarchical and timed Petri Nets, which respectively organizes a system into modules and adds timers to the transitions. This topic is detailed in Jensen and Kristensen (2009), with details on the types of CPNs and formal definitions.

Jensen *et al.* (2007) describe a detailed example of using the CPN Tools to model and validate a network protocol. In this example, simulations and a space state analysis are performed to verify a CPN model. Using simulations, it is possible to identify errors, but it is not possible to prove that they do not exist, because it is not feasible to check all the existing scenarios. Thus, to validate a model, it is necessary to analyze the state space of a CPN. The state space is a directed graph representing all possible execution flows, where each arc indicates a system transition, changing the marking of the CPN. In addition to state space, CPN Tools generates a report with much more information, including, for example, the possible markings for each place.

It is essential to comment on the state space problem that occurs when it is not possible to generate a finite state space, making it unfeasible to validate a model. In this context, Seifi (2014) describes strategies to facilitate the modeling and prevent the state space problem. It is possible to:

- define places to indicate errors and thus interrupt the creation of the state space upon reaching an unexpected marking;
- build parameterized models, allowing modifications of the modeling through of constants;
- control the execution by using tokens to enable or disable transitions; and
- manipulate the tokens that can be stored in each place to prevent that infinite markings represent the same state, containing, for example, duplicate tokens.

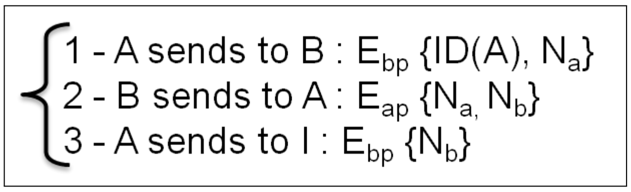
In order to verify that a modeling system satisfies a specific property and validates a CPN, a model check must be performed, using functions in CPN ML (JENSEN *et al.*, 2007). During the validation, the functions in CPN ML are analyzed in all the system markings, observing the tokens stored in each place to detect some flaw. The path executed until the flaw can be parsed to identify the cause of an error and propose one correction.

System validation can also be performed using temporal logics, such as Linear Temporal Logic (LTL) and Computational Tree Logic (CTL) (CLARKE *et al.*, 1986). By using temporal logic, the behavior of a system along the execution paths is verified. It is possible, for example, to define logical formulas to check if all paths meet a given condition or if there is any path where a condition is satisfied. Thus, a logical formula could be developed, based on the messages sent, to detect the same error previously exposed. Cheng *et al.* (1997) present the use of CTL formulas for the analysis of the state space of a CPN model.

2.4.2 Modeling and validation of NS protocol

Lowe (1995) reported a security failure of the NS protocol. Figure 5 describes the messages exchanged in the original protocol to provide mutual authentication of entities *A* and *B*. In this protocol, the ciphered messages are represented by the tags E_{ap} and E_{bp}, indicating the cryptography with the public keys of *A* or *B* respectively. Thus, only *A* can decipher the messages sent by *B*, and only *B* can decipher the messages sent by *A*. The other elements in the messages are the identity of *A* (*ID(A)*) and the random nonces N_a and N_b. After these messages, *A* authenticates *B* and vice versa.

Figure 5 – NS protocol

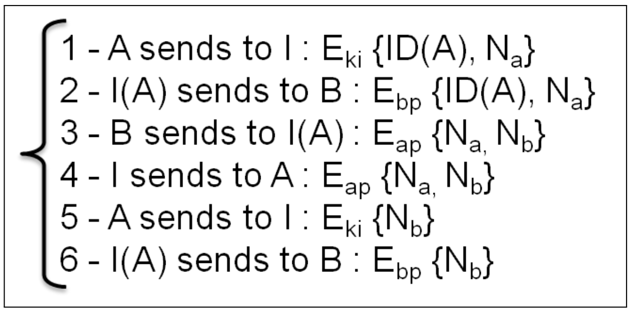


1 - A sends to B : E_{bp} {ID(A), N_a}
2 - B sends to A : E_{ap} {$N_{a,}$ N_b}
3 - A sends to I : E_{bp} {N_b}

Source – (SEIFI, 2014).

In order to demonstrate that the original protocol is insecure, it is enough to show an attack in which an intruder impersonates one agent of the communication, violating the authenticity property, as shown in Figure 6. After these messages, *B* authenticates the intruder *I*, but *B* thinks the communication is with *A*.

Figure 6 – NS protocol attack



1 - A sends to I : E_{ki} {ID(A), N_a}
2 - I(A) sends to B : E_{bp} {ID(A), N_a}
3 - B sends to I(A) : E_{ap} {$N_{a,}$ N_b}
4 - I sends to A : E_{ap} {$N_{a,}$ N_b}
5 - A sends to I : E_{ki} {N_b}
6 - I(A) sends to B : E_{bp} {N_b}

Source – (SEIFI, 2014).

Although it is possible to demonstrate the insecurity of a protocol describing an attack, it is necessary a formal validation to prove that a protocol is really secure. In this context, Seifi (2014) details the use of CPNs to validate a security protocol, using the NS protocol as example. The analysis of the space state can be used to prove if some property is violated in any reachable state, demonstrating the security or not of a protocol.

Seifi (2014) proposes a methodology for the modeling of security protocols using CPNs. In this methodology, a hierarchical CPN is designed based on the messages of the protocol, using a top-down approach. Then, a preliminary model is evaluated without observing malicious behaviors. Next, the behaviors of the attackers are modeled.

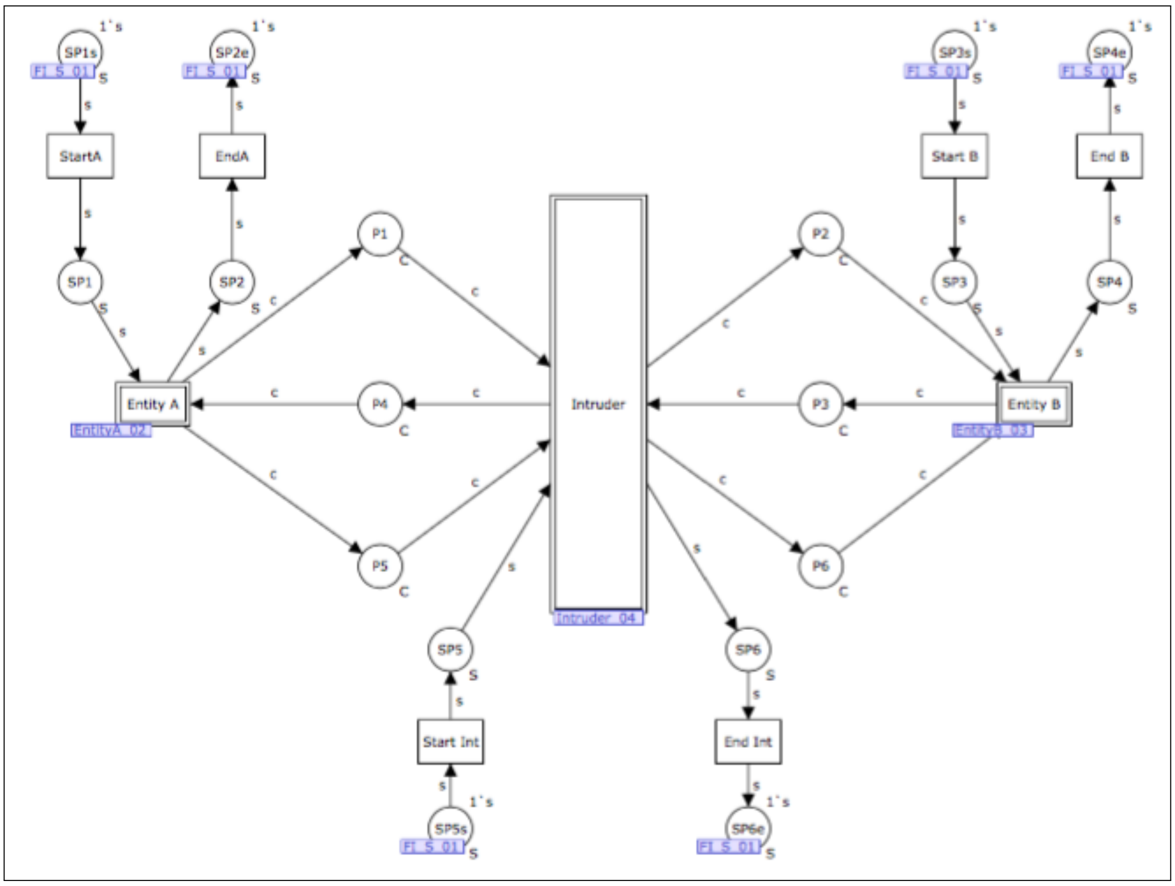
Figure 7 presents an overview of the NS protocol modelled with CPN. In summary, this CPN shows the communication between the entities A and B in accordance with the protocol, in which the tokens pass through: i) P1 and P2 in the first message; ii) P3 and P4 in the second message; and iii) P5 and P6 in the last message (see Figure 5). The SPs places are related to the transitions to initiate and terminate the actions inherent in each entity, including the intruder the is responsible to perform attacks during the protocol trying to break its security. The behaviour of each entity is described in sub-modules and the complete description of this modeling is available on Seifi (2014).

In the security protocols, the behavior of an attacker is traditionally defined by the Dolev-Yao model (DOLEV; YAO, 1983), in which the attacker can: i) copy or block messages; and ii) modify or create messages, limited by cryptographic restrictions. The attacker cannot break the encryption, but if he/she has the keys, he/she can decrypt messages and store them in a database, along with captured messages.

In this context, the security analysis of the NS protocol aims to demonstrate if there are violations of the authenticity property, based on actions that can be performed by an attacker. For this purpose, the authenticity property is translated into logical formulas or functions in CPN ML to validate the CPN model. For example, Seifi (2014) defines a function to verify if it is possible that *B* authenticates *A* even when *A* is not communicating with *B*. This function returns the markings that indicate authentication violations. Using these markings, it is possible to create the sequence of messages that generate the violation.

It is interesting to mention that cryptographic algorithms are usually considered secure during an evaluation of security protocols. The formal methods are not suitable to assess the security of these algorithms that are analyzed using cryptanalysis' methods. The NS protocol

Figure 7 – Overview of the NS protocol modelled with CPN



Source – (SEIFI, 2014).

uses an asymmetric cipher and its security is not discussed in the validation of this protocol. Due to the use of standard cryptographic algorithms, the assumption of their security is valid in this thesis.

2.5 Summary

This chapter presented the concepts and requirements of a secure cloud storage solution that focuses on guarantees of security properties and data sharing. Thus, besides the stakeholders, the required features are enumerated, including the monitoring and auditing mechanisms that are used to verify these properties. Based on these mechanisms, the provider can prove that no violations occurred. In addition, the access control mechanism is essential to manage the keys according to the users' permissions and to avoid unauthorized access.

The existing threats were also shown in this chapter, detailing the attacks that violate the security properties. The analyzed attacks are related mainly to the malicious behaviors of the provider or the broker. These attacks cannot be avoided and must be detected by monitoring and auditing mechanisms. Within the scope of this thesis, violations of confidentiality, integrity, freshness and write-serializability are identified.

Last, the use of CPNs to verify the violations of security properties in protocols is described. This formal method is used to validate the mechanisms proposed in this thesis, checking if all violation scenarios are detected.

3 RELATED WORK

The literature reveals solutions that address security issues in cloud storage, especially related to data corruption, loss and leakage. The existing solutions protect the system against these threats, avoiding or detecting an attack. This chapter focuses on related work that proposes mechanisms to detect violations of the security properties addressed in this thesis. The related work is part of the second activity of the proposed methodology (see Section 1.5).

So, this chapter is organized as follows: Sections 3.1 and 3.2 discuss how the security properties are verified by related work; next, Section 3.3 exposes the issues related to real-time detection and collusion attacks; discussions are guided by the attacks described in Section 2.3, identifying the limitations of the related work in detecting the violations; and, finally, this chapter is summarized in Section 3.4.

3.1 Confidentiality and Integrity

Confidentiality and integrity are properties related, respectively, to data leakage and corruption. Security solutions commonly specify the use of symmetric ciphers to provide confidentiality and schemes of digital signature or MAC (Message Authentication Code) for the verification of integrity (STALLINGS, 2016). In a data sharing environment, the security mechanisms are designed to define the users' permissions and distribute the encryption/reading keys only to authorized users (THILAKANATHAN *et al.*, 2014). In this context, unauthorized entities cannot read a file even if they have access to the encrypted file, preventing the data leakage. Therefore, the related work found in this research uses encryption mechanisms to ensure confidentiality, not being necessary to verify the occurrence of confidentiality violations (POPA *et al.*, 2011).

On the other hand, Kallahalla *et al.* (2003) reveal the possibility of a revoked user accessing a file. Although a new encryption key is generated when the permissions are updated, this user can read a file using an old key if it is not re-encrypted yet with the new key. Thus, it is not possible to ensure confidentiality only using encryption mechanisms, being essential to verify if each file access is really authorized. It is timely to inform that Kallahalla *et al.* (2003) discuss attacks performed by revoked users without detecting them in the proposed solution.

Besides the encryption key, a signing/writing key is used by existing solutions to sign each file so that only the authorized users can write files. Although the malicious provider can perform an illegal writing requested by an unauthorized user, it is not possible the generate a valid signature of this writing without knowledge about the writing key. Thus, the users can detect data corruption when reading a file, checking the integrity of this file using its public verification key. The existing solutions enable the detection of integrity violations using this approach.

However, similarly to the confidentiality violations, it is necessary to analyze the illegal writings performed by revoked users. Although a revoked user cannot access the new keys, he/she can sign the new version of a file, using an old writing key. In this scenario, the violation is detected if the integrity is verified using the current verification key, but the file is considered authentic when checking with an old verification key.

The related work addresses the generation of new keys when updating the permissions, but does not analyze the possibility of using a verification key that is no longer valid in the integrity check. In summary, for not considering transactions performed by revoked users, the existing solutions partially detect integrity violations and do not identify confidentiality violations.

3.2 Freshness and Write-serializability

Freshness and write-serializability are properties related to data loss. After a data loss, a malicious provider can, for example, recover old versions of the files from a backup, without reporting this incident to the users. Existing solutions detect this type of attack through the verification of the order of the file versions throughout the transactions. Normally, the violations are identified in auditing time, analyzing the log of the transactions (POPA *et al.*, 2011) (HWANG *et al.*, 2016).

In CloudProof (POPA *et al.*, 2011), the attestations are used to record each read or write transaction, and these attestations are chained so that it is possible to sort the attestations during auditing. The attestations are sent by the users for auditing and contain a field that indicates the version of the file accessed. Thus, it is possible to check if the accessed version in each transaction is the one expected. In case of violation, the type of the transaction reveals a freshness or write-serializability violation. Due to the signature of the attestations, it is not possible to deny or accuse the occurrence of a violation falsely. The auditing is periodically carried out, and some blocks can be randomly selected for auditing, reducing the overhead. Albeshri *et al.* (2012a) and Tiwari and Gangadharan (2015a) extended the CloudProof, preserving its basic structure.

In the solution proposed by Hwang *et al.* (2016), the attestations are stored by the untrusted provider and not by the users. In this case, a malicious provider can execute a rollback attack, discarding the attestations regarding the lost versions of a file. The users keep the last sequence number that is also included in the attestations so that the absence of any sequence number indicates a violation. Although some users can detect the violations when performing a transaction, it is not possible to ensure this detection as soon as an attack occurs. In addition, the authors discuss the disposal of old attestations and concurrent access.

However, the security evaluation of these solutions is based on the informal discussion of the presented theorems, and the absence of a formal evaluation may result in security flaws. In this context, it is essential to analyze an existing solution formally as specified in the third activity of the methodology used in this thesis (see Section 1.5). Then, CloudProof was modeled and validated, using CPNs, to identify scenarios in which security violations are not detected (CARVALHO *et al.*, 2016a). These violations exist because the detection of violations is based only on the chain of attestations, without considering the stored data.

For example, a malicious provider cannot report the loss of a file. If after the loss, the following transaction is a reading transaction, the violation is detected. However, the violation is not identified when executing a writing transaction immediately after this loss. This occurs if no information about the chain of attestation has been lost and because the provider can store the new version of the file, properly adding that transaction to the chain of attestations. Then, it is necessary to verify not only the chain of attestations, but also the hold of the file before each writing.

In this context, the related work fails in detecting all scenarios of write-serializability violations due to the focus in verifying only the chain of attestations (POPA *et al.*, 2011) (ALBESHRI *et al.*, 2012b) (HWANG *et al.*, 2014) (TIWARI; GANGADHARAN, 2015a) (HWANG *et al.*, 2016). The solution proposed by Jin *et al.* (2018) is another study analyzed in this thesis, but it does not address the write-serializability violations.

3.3 Real-time detection and Collusion attacks

The malicious actions enumerated in Section 2.3 are performed by providers so that it is not possible to avoid an attack. Thus, the security mechanisms used in the related work focus on the detection of violations after an attack. In the case of an attack, the moment of the violation detection is an important aspect to be observed, avoiding the use of an outdated or corrupted file.

The solutions proposed by Hwang *et al.* (2014) and Jin *et al.* (2018) are more effective, detecting the violations during the execution of transactions. Although the detection is on the next transaction, the authors use the terms *real-time* (HWANG *et al.*, 2014) and *instantaneous* (JIN *et al.*, 2018) to indicate the detection of violation as soon as possible. Then, in this thesis, the term real-time is used to denote the verification of properties in monitoring time, during the execution of transactions. In other analyzed studies, the violation detection is in auditing time.

Hwang *et al.* (2014) specify the use of a trustworthy synchronization server to inform users of the current state of the provider, enabling the violation detection in real-time. Without this third party, the users do not have enough information to identify if, for example, the file sent by the provider is old.

The assumption of an honest server is acceptable when it is deployed in a private cloud. However, if this server is untrustworthy, it is essential to analyze the collusion attacks that: i) allow data leakage; or ii) bypass the detection of data loss and corruption in real-time. Therefore, the auditing is indispensable to detect all violations. The auditing can also be performed to solve any contestation due to the non-repudiation of cloud transactions (HWANG *et al.*, 2016).

In this context, the management of the transactions by a third party enables, if this party is honest, the real-time detection of data loss or, otherwise, the occurrence of collusion attacks. For this reason, the collusion attacks are unfeasible in solutions that identify only integrity violations in real-time.

The exception is the solution proposed by Jin *et al.* (2018), in which the freshness violations are detected in real-time, but there is no third party involved in the communication. In an informal analysis, the authors declare that *"if the cloud sends a stale data block..., it is impossible for the stale data to pass the root authentication check using the latest root signing key kept by the client"*. However, the authors do not prove that the key held by the users is really the last one. Hwang *et al.* (2014) use a similar approach based on hash trees in which the reliance on a third party is considered essential to ensure freshness verification in real-time.

3.4 Comparison of Related Work

Table 3 presents a comparison between the solutions, discussed in this chapter, for the verification of the following security properties in a cloud storage environment: integrity, freshness, write-serializability and confidentiality. This table highlights the existing gaps that are related to the detection of violation of these security properties as well as if the detection occurs in real-time. Moreover, it also shows the gaps related to the identification of collusion attacks.

In short, Table 3 summarizes the discussion of Section 3.1 on the fact that the existing solutions do not detect confidentiality violations ("No" in the table) and fail in the identification of integrity violations, resulting from transactions performed by revoked users ("Partially" in the table). As shown in Section 3.2, the related work do not properly verify the write-serializability that should also analyse the stored file ("Partially" and "No" in the table). Section 3.3 exposes that the collusion attacks are inherent to the existence of a third party, responsible to manage the cloud transactions. Thus, in the table, the collusions attacks is considered "Unfeasible" in studies that do not have this third party. A single study includes the third party (HWANG *et al.*, 2014), but the collusion attacks are "Not detected", because this entity is considered trustworthy. Last, the integrity violations is detected, in real-time, when reading a file by existing solutions, but only two works propose the violation detection of other properties in real-time (see Section 3.3).

Table 3 – Comparison of related work

Work	Security Properties				Real-time Detection	Collusion Attacks
	I	F	W	C		
(POPA et al., 2011)	Partially	Yes	Partially	No	I	Unfeasible
(ALBESHRI et al., 2012)	Partially	Yes	Partially	No	I	Unfeasible
(HWANG et al., 2014)	Partially	Yes	Partially	No	I, F, W	Not detected
(TIWARI; GANGADHARAN, 2015a)	Partially	Yes	Partially	No	I	Unfeasible
(HWANG et al., 2016)	Partially	Yes	Partially	No	I	Unfeasible
(JIN et al., 2018)	Partially	Yes	No	No	I, F	Unfeasible

I - Integrity; F - Freshness; W - Write-serializability; C - Confidentiality

Although there are other security properties, only studies that focus on the properties addressed in this thesis are presented. For example, Albeshri *et al.* (2012b) and Jin *et al.* (2018) also address the verification, in auditing, of the retrievability of files scarcely accessed.

Lastly, it is important to mention the evaluation of the related work presented in this chapter. The trend in literature is the use of standalone hosts to evaluate the cost of the proposed algorithms, together with an informal security discussion. The single exception is Cloudproof, which was deployed in the Azure infrastructure (POPA *et al.*, 2011). However, the performed experiments do not indicate the absence of security flaws.

3.5 Summary

This chapter summarized the existing work related to this thesis subject and their limitations, enumerating attacks that are not detected by them. The described analysis reveals the necessity to design a mechanism that: i) includes the verification of permissions to identify the transactions executed by revoked users; ii) checks the write-serializability properly; and iii) detects collusion attacks.

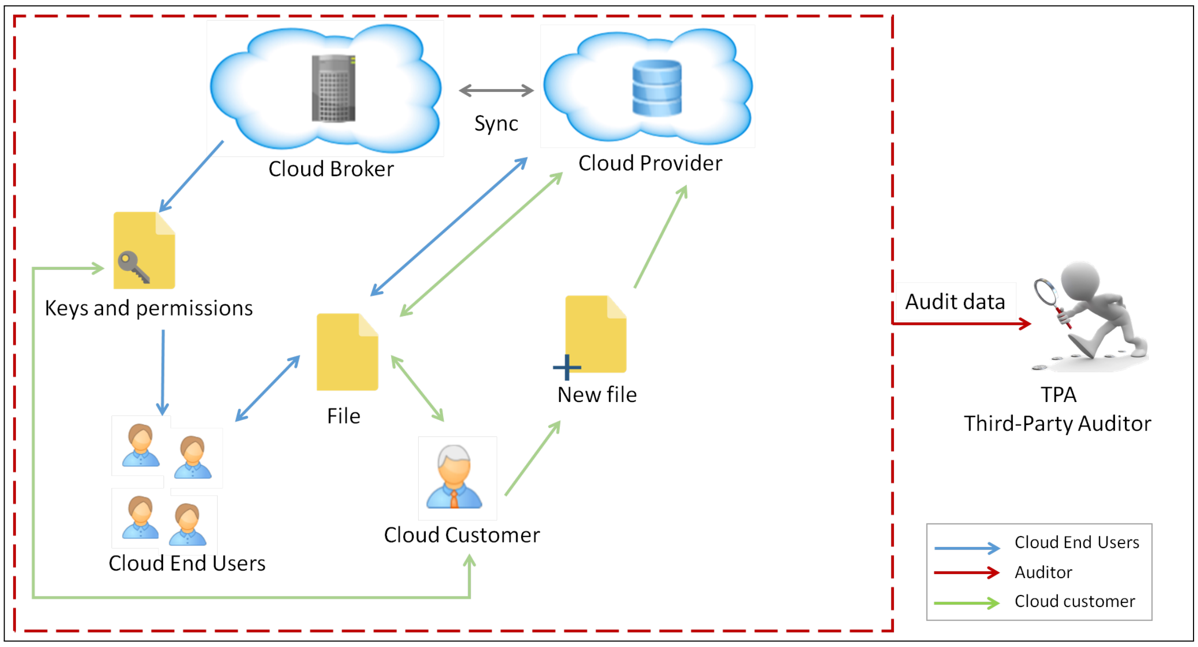
4 SCUDO DESCRIPTION

In this chapter, SCUDO (Secure CloUd storage service for Detecting viOlations of security properties) is described. In a nutshell, this storage service protects the users' data, ensuring security properties based on monitoring and auditing mechanisms. These mechanisms are combined with an access control mechanism to allow data sharing. This chapter is then divided as follows. First, Section 4.1 presents an overview of the SCUDO and then the security mechanisms considered in this work are detailed. The access control mechanism is presented in Section 4.2, the protocol to perform cloud transactions and to detect violations is described in Section 4.3, and the auditing process is exhibited in Section 4.4.

4.1 Objective

The proposed storage service works similarly to existing storage services, in which the users read and write files in a cloud provider, and the customer defines permissions for each file. This service is managed by a broker to enable real-time detection of violations and control concurrent transactions. Besides, the transactions can be audited by a TPA (Third Party Auditor) to demonstrate the security. Figure 8 presents the communication links between the entities and the exchanged artifacts.

Figure 8 – SCUDO communication model



Source – The author.

SCUDO aims to detect all attacks described in Section 2.3 in real-time if the broker is honest. Otherwise, TPA will identify the violations in auditing by examining the transaction log. Therefore, SCUDO improves the violation detection in comparison with related work, addressing other attack scenarios, such as collusion attacks.

The monitoring of security properties is performed during the execution of cloud transactions. So, the protocol for reading/writing files and permission updating includes steps to verify these properties.

Before detailing the auditing and monitoring mechanisms, it is necessary to describe the access control used by SCUDO, which is responsible not only for the setting of permissions, but also for data encryption and key management.

4.2 Access control

The use of an access control mechanism is essential to allow data sharing and prevent data leakage. Proxy Re-Encryption (PRE), Attribute-Based Encryption (ABE) and Access Control Lists (ACLs) are the approaches used to design access control mechanisms (THILAKANATHAN *et al.*, 2014). A proxy, without obtaining the plaintext, can use a PRE scheme to translate a ciphertext into another ciphertext that is deciphered by the authorized receiver (TIWARI; GANGADHARAN, 2015b). On the other hand, in the ABE approach, the access is allowed if the attributes of the data and the user match in accordance with an access control policy (TASSANAVIBOON; GONG, 2011). Last, an ACL is a list indicating permissions to access a resource for each user so that the provider should only perform the transactions requested by authorized users.

In addition to the use of ACLs, the encryption is essential to protect the access and the modification of the files stored in a cloud provider (TIWARI; GANGADHARAN, 2015b). In this context, the key management is a fundamental feature to avoid unauthorized users from obtaining keys.

4.2.1 ACLs and Encryption

ACLs have been used to specify users' permissions with solutions that also focus on violation detection (POPA *et al.*, 2011) (JIN *et al.*, 2018). Public providers also use ACLs in the access control mechanism (*e.g.*, Amazon S3 (AMAZON, 2018b)). Due to these aspects, ACLs are used, in this thesis, to specify users' permissions.

Each file has an ACL that contains a list of users with their permissions. Generally, the users have read or read/write permissions, and only the cloud customer can change the permissions, granting or revoking access. For efficiency purposes, it is possible to use one single ACL to authorize the access to a group of files (KALLAHALLA *et al.*, 2003).

In addition to setting permissions, it is essential to use cryptography algorithms to make effective the access control so that providers do not have access to plaintext (TIWARI; GANGADHARAN, 2015b). A symmetric cipher is used for data confidentiality so that only authorized users can encrypt or decrypt the file, using the secret key (R). Thus, not even the provider can read the stored files. Data integrity is ensured based on the digital signature in which the pair of keys (W and W_v) is used, respectively, to sign the file and verify its signature. These keys are generated by the process defined by the asymmetric cipher used for digital signature. Thus, it is possible to verify if a file was created or modified by an unauthorized entity.

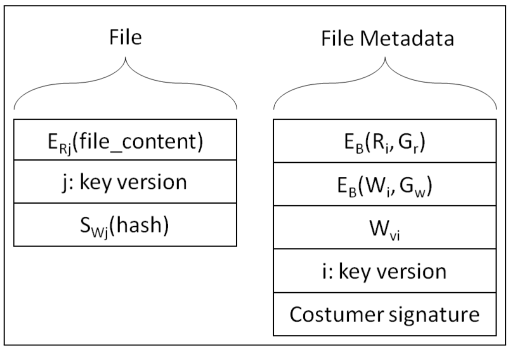
The security of the cryptographic primitives used by cloud storage solutions is attested by the scientific community and is outside the scope of this thesis. The AES (Advanced Encryption Standard) and RSA (Rivest-Shamir-Adleman) are, respectively, the symmetric and asymmetric cipher used by SCUDO. It is recommended to use different keys for each file and change them when the permissions are updated, following the guideline to limit the amount of data encrypted with a single key (BARKER *et al.*, 2012) and avoiding that revoked users can access a file (KALLAHALLA *et al.*, 2003). The keys must be accessed in accordance with the users' permissions, enabling granular access control.

4.2.2 Key Management

In this context, the key management is essential to distribute the keys only to authorized users and update the keys whenever necessary. In a costly and straightforward solution, while W_v is kept public, R and W can be encrypted with the public key of each authorized user so that they are accessed only by them. Due to the inefficiency of this approach, Broadcast Encryption has been used (POPA *et al.*, 2011) (JIN *et al.*, 2018). The key and the group of authorized users (G) are the inputs of a broadcast encryption scheme (E_B), generating a ciphertext that can be deciphered only by users in G. Boneh *et al.* (2005) detail a scheme that enables broadcast encryption, allowing the decryption of keys only by authorized users.

Figure 9 illustrates the layout of each stored file and its metadata, detailing how each key is used. The users with read or read/write permissions belong to the set G_r and can decipher the $E_B(R_i, G_r)$, extracting the key used to encrypt/decrypt the file. W_i can be obtained by users with read/write permissions (*i.e.*, members of the set G_w). Thus, authorized users can modify a file, encrypt it and generate a valid signature. The broker stores the metadata for each file and sends it to authorized users during the transactions.

Figure 9 – File and file metadata layout



Source – The author.

In addition, there are different versions of the reading and writing keys, as indicated by the indexes *i* and *j*. The key version used to encrypt the file can differ from the key indicated in the metadata due to the lazy revocation approach. This approach is frequently adopted to reduce the overhead when permissions are revoked (JIANG *et al.*, 2014) (TASSANAVIBOON; GONG, 2011). With lazy revocation, the keys are updated, and the file is re-encrypted only in the next writing transaction. When the permissions are updated, the key version (index *i*) in metadata is also updated. However, the file is not re-encrypted so that its key version (index *j*) remains unchanged.

The Key Rotation scheme is used to simplify the keys updating, when the permissions are changed and enables the lazy revocation (KALLAHALLA *et al.*, 2003). In this scheme, only the customer can create the next key version and signs the new metadata when modifying the permissions. He/she generates a sequence of keys from an initial/master key and his/her private key while the users can extract any previous version using the public key of the customer. Thus, after deciphering the keys from the file metadata, authorized users can obtain the correct key (index *j*) used to cipher or sign a file from the current version (index *i*). It is interesting to mention this scheme can create keys to be used by symmetric and asymmetric ciphers.

The security of the Broadcast Encryption and Key Rotation schemes have been already discussed in the literature, showing that no unauthorized user can obtain the credentials to access the files (BONEH *et al.*, 2005). On the other hand, it is necessary to analyze the occurrence of data leakage when revoked users read files using old keys (KALLAHALLA *et al.*, 2003). In SCUDO, the data leakage is detected in the following scenarios: i) a provider sends an old version of the file from, for example, a backup; and ii) the file was not re-encrypt yet due to the lazy revocation.

Although it is not possible to avoid the data leakage with the immediate revocation, there is a loss of security inherent to the lazy revocation. The immediate re-encryption hinders the data leakage, preventing the occurrence of the second scenario aforementioned. An example of this scenario is when a malicious broker authorizes a revoked user to read a file. The reading is possible if this file has not been re-encrypted yet and even if the provider is honest.

In this context, despite the higher cost of updating permissions due to the re-encryption, the lazy revocation is not applied by SCUDO. Therefore, the use of a Key Rotation scheme becomes unnecessary so that any user (and not only the customer) can generate new keys to cipher or sign a file. This approach offers greater security by allowing the keys' updating whenever a file is modified, following a security recommendation to avoid the reuse of keys (VAUDENAY; VUAGNOUX, 2007). In addition, it is possible to allow the writing of new files and/or the updating of permissions by other users.

Section 2.3 details other attacks that cannot be avoided, characterized by malicious behaviors of providers and/or brokers. The detection of violations resulting from these attacks is the focus of this thesis, and the rest of this chapter describes how SCUDO verifies the security properties during the monitoring and auditing.

4.3 Cloud Transactions and Monitoring

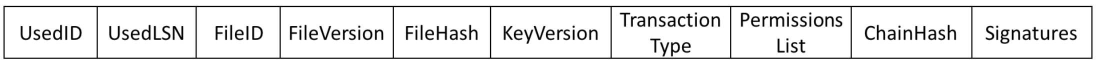
This section specifies the protocols to read, write and update the permissions of a file. If a trustworthy broker is used, the security violations can be detected in real-time, during these transactions. The verification of the properties is based on log analysis and each log entry is called attestation.

4.3.1 Attestation

The broker manages the transactions, being responsible for sending the attestation of the last transaction. Using this attestation, a user verifies the integrity and freshness of the read file or prepares a writing request that complies with the write-serializability. It is important to mention that the holding of the users' privacy is guaranteed, because the attestations as well as the metadata do not have any sensible information available to malicious entities.

Figure 10 details the structure of an attestation that is composed of the following elements: **UserID**, **UserLSN**, **FileID**, **FileVersion**, **FileHash**, **KeyVersion**, **TransactionType**, **PermissionsList**, **ChainHash** and **Signatures**.

Figure 10 – Structure of an attestation



UsedID	UsedLSN	FileID	FileVersion	FileHash	KeyVersion	Transaction Type	Permissions List	ChainHash	Signatures

Source – The author.

The **UserLSN** represents the last sequence number used by each user, enabling to them detect replay and rollback attacks. The **FileVersion** indicates the version of a file, being helpful to verify the freshness and write-serializability, while the **FileHash** contains the file signature that is used to check the integrity. The **KeyVersion** allows a user to derive the correct key using a Key Rotation scheme. The **TransactionType** indicates the execution of a transaction for reading, writing, or updating the permissions of a file. The **PermissionsList** is a field inherent only of transactions for updating permissions to report their changes. 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.

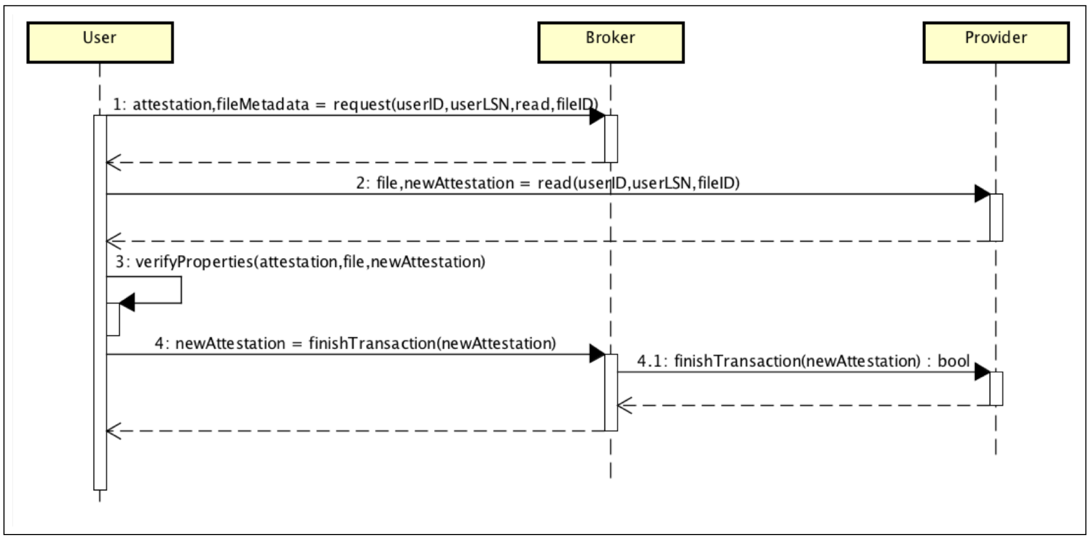
Due to the **Signatures** of an attestation, it is not possible to change its content in such a way as to report, for example, a different **FileHash**, allowing to bypass the integrity verification. Similarly, it is not feasible to inform that another user performed a specific transaction. In addition, the building of the attestations' chain enables to prove the execution of transactions by revoked users or the sending of outdated files.

4.3.2 Reading and Writing

The users and also the cloud customer read and write files following the protocols described in this section. The broker manages the transactions by storing the last attestation of each file and sending them to users in order to verify the security properties. The monitoring of the properties is performed during cloud transactions to detect violations in real-time.

Although there is a storage overhead, the monitoring is not possible if the broker does not store one attestation per file. The users require the last attestation of the accessed file to compare with the received information from the provider. The Sequence Diagram[1] of the protocol to read a file is detailed in Figure 11, and Transport Layer Security (TLS) is used to protect the communication between the stakeholders.

Figure 11 – Reading a file


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

Source – The author.

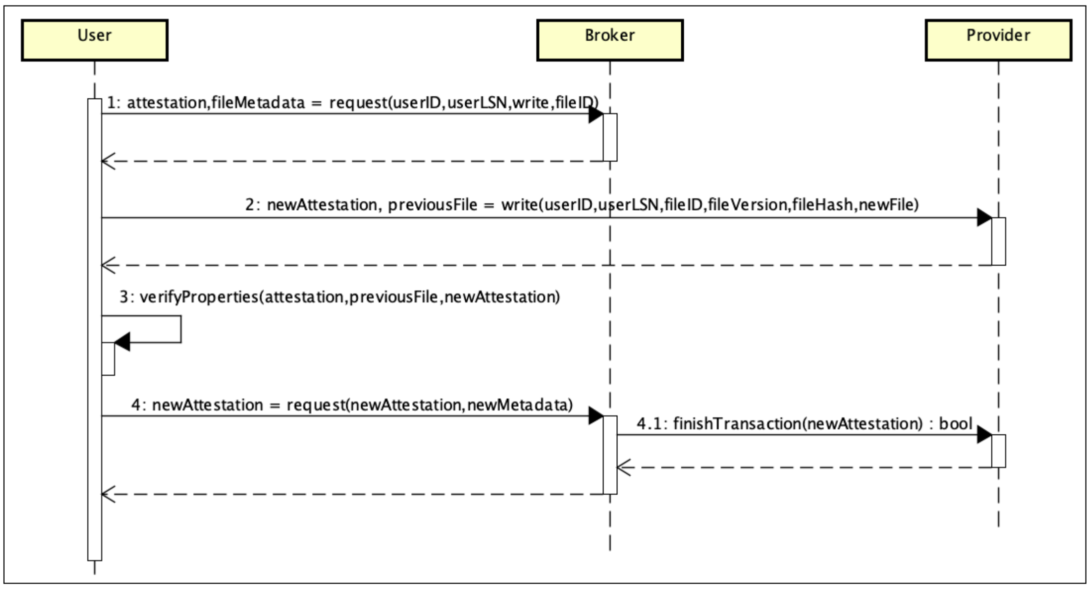
Before a user requests the reading of a file, a user sends a message to the broker in order to receive the last attestation and the file metadata. Before answering a request, the broker verifies the permissions, avoiding sending information to unauthorized users. Next, the user requests the file to the provider. The provider sends the file and the new attestation signed by it. Then, the user extracts the encryption key and the verification key in accordance with the **KeyVersion**. While the encryption key is used to decipher the file, the verification key is used to check the file integrity. The signature must be the one indicated by the broker in the **FileHash** field, also proving the freshness of the received file. In addition, the **FileVersion** of the attestations of the last and current transactions must be the same. 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 while the provider stores all attestations for auditing.

Similarly, the broker sends to an authorized user the last attestation and metadata for a writing transaction (see Figure 12). During a writing, a user deciphers the encryption and signature keys. The encryption key is used to cipher the file while the signature key is to sign

[1] The Sequence Diagram is one diagram of the UML (Unified Modeling Language) (RUMBAUGH *et al.*, 2004).

it. The encrypted file is stored by the provider and its signature is placed in the **FileHash** field of the new attestation. The provider also increments the **FileVersion** for write-serializability purposes before signing this attestation. As in reading, this attestation is also signed by the user and the broker.

Figure 12 – Writing a file


User
Broker
Provider
1: attestation,fileMetadata = request(userID,userLSN,write,fileID)
2: newAttestation, previousFile = write(userID,userLSN,fileID,fileVersion,fileHash,newFile)
3: verifyProperties(attestation,previousFile,newAttestation)
4: newAttestation = request(newAttestation,newMetadata)
4.1: finishTransaction(newAttestation) : bool

Source – The author.

Only these steps are not enough to detect all attacks that result in write-serializability violations. For example, a malicious provider may have lost a file and still write a new version of this file using the expected **FileVersion** based on the last attestation. So, it is necessary to verify if the file is really stored by the provider before a writing and its version as discussed in Section 3.2. The verification of the ownership of a file before each writing is essential to enable the violation detection. Then, in order to check if a file was not lost, the provider sends back the previous version of the file when performing a writing transaction (see Figure 12).

This sending raises the overhead of the writing protocol, especially when the file is large. However, the efficiency of this protocol can be improved, using a PoR (Proof of Retrievability) scheme (WANG *et al.*, 2013). The PoR schemes are out of the scope of this thesis, but their security is discussed by existing solutions (ALBESHRI *et al.*, 2012b) (TIWARI; GANGADHARAN, 2015a) (JIN *et al.*, 2018). It is clear that the writing of new files does not include the verification of ownership. In SCUDO, similarly to related work, only the cloud customer can write new files.

4.3.3 Updating Permissions

Besides the reading and writing transactions, it is necessary to specify the protocol for updating permissions. This type of transaction is essential in a data sharing environment, enabling to grant or revoke privileges to users. In SCUDO, similarly to sending new files, only the cloud customer can update the permissions.

It is important to mention that the attestation of a permission update transaction must also be chained with other transactions of a file. This attestation includes the field **Permission-sList**, and each element of this list indicates the **UserID** and his/her new permission. In order to reduce the size of this list, only the users whose permissions have been changed are added to the list.

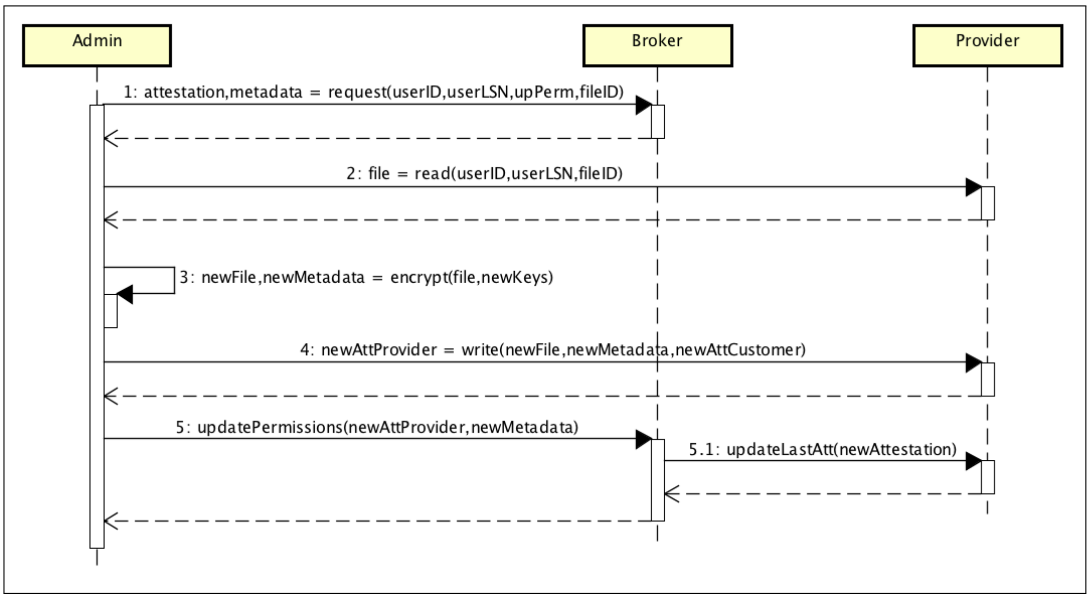
Figure 13 details the protocol to change the permissions. This protocol does not follow the lazy revocation approach, and the customer re-encrypts the file when the permissions are updated. Before encryption, the customer receives the last attestation and metadata from the broker and the file from the provider. The integrity and freshness are verified and, then, the customer creates the new metadata with new keys and encrypts the file. Next, the file is sent to the provider that sends back the attestation. If no error occurs, the customer and the broker sign the attestation, finishing the transaction. As an optimization, the re-encryption is only performed if some user loses the privileges. Thus, it is possible to remove the cost to encrypt the file and to transmit the file from and to the provider.

Malicious actions of the provider result in violations that are detected by the monitoring of security properties. After an attack (*e.g.*, in next transaction), an inconsistency between the last attestation and the stored file is identified. The attestation sent by the broker reports the current/expected state of the file so that a user can verify the integrity and freshness of the stored file. Therefore, a malicious provider cannot bypass the violation detection, hiding an unauthorized writing or sending a previous version of a lost file.

However, the user must be sure that the received attestation is really the last one, making indispensable the participation of an honest broker to manage the transactions. In the case of violations, the transaction is canceled, and 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 resulting from the use of corrupted data.

On the other hand, some attacks cannot be detected by monitoring due to the

Figure 13 – Updating file permissions


Admin
Broker
Provider
1: attestation,metadata = request(userID,userLSN,upPerm,fileID)
2: file = read(userID,userLSN,fileID)
3: newFile,newMetadata = encrypt(file,newKeys)
4: newAttProvider = write(newFile,newMetadata,newAttCustomer)
5: updatePermissions(newAttProvider,newMetadata)
5.1: updateLastAtt(newAttestation)

Source – The author.

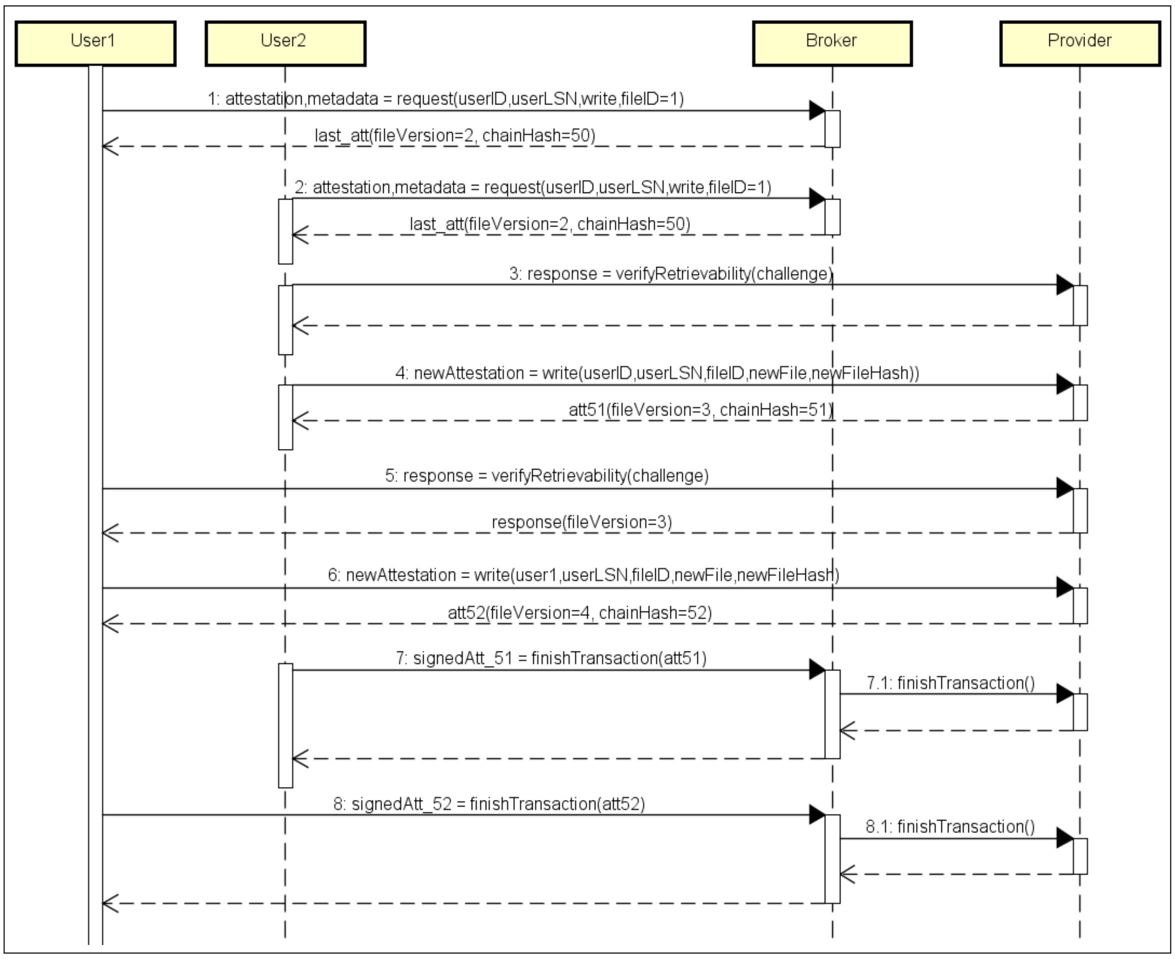
malicious actions of the broker. In these attacks, the broker usually colludes with the provider and even with revoked users to deceive the violation detection. Therefore, it is necessary to audit the transactions to identify all violation scenarios or to demonstrate the security of the provided service. The auditing is also performed when detecting a violation in order to avoid that a user falsely accuses the cloud provider, solving any conflict. Before detailing the auditing mechanism, it is necessary to analyze the execution of concurrent transactions.

4.3.4 Concurrent Transactions

The management of concurrent transactions allows simultaneous access from different users. In this section, the simultaneous access to the same file is discussed. The users do not know about the others' concurrent transactions and can report a violation identifying inconsistencies between the attestations of the last and current transactions. Figure 14 presents an example in which the transactions end in a different order from which they started.

In this example, *User1* starts a writing transaction and receives the attestation referring to the file and indicating the *ChainHash* of the last transaction (the transaction number is 50). Before continuing the transaction, *User2* normally performs the transaction 51, writing a new version of this file. In this case, *User1* verifies the ownership of the third version of the file, but he/she has the keys and tags referring to the second version. Besides, *User1* receives the attestation of the transaction 52 (with **FileVersion** = 4) and not 51 (with **FileVersion** = 3) as

Figure 14 – Concurrent Transactions Example


User1
User2
Broker
Provider
1: attestation,metadata = request(userID,userLSN,write,fileID=1)
last_att(fileVersion=2, chainHash=50)
2: attestation,metadata = request(userID,userLSN,write,fileID=1)
last_att(fileVersion=2, chainHash=50)
3: response = verifyRetrievability(challenge)
4: newAttestation = write(userID,userLSN,fileID,newFile,newFileHash))
att51(fileVersion=3, chainHash=51)
5: response = verifyRetrievability(challenge)
response(fileVersion=3)
6: newAttestation = write(user1,userLSN,fileID,newFile,newFileHash)
att52(fileVersion=4, chainHash=52)
7: signedAtt_51 = finishTransaction(att51)
7.1: finishTransaction()
8: signedAtt_52 = finishTransaction(att52)
8.1: finishTransaction()

Source – The author.

expected by him/her. It is possible to delegate the verification to the honest broker that knows about the second transaction and can analyze the chain of transactions properly.

In this example, the provider sends the attestation of the transaction 52 before receiving the confirmation of the transaction 51. This behavior is essential to make feasible the concurrent access, avoiding the existence of two transactions with the same number. However, there is no guarantee that all transactions will be finished properly. If a network failure occurs during the step seven and only the transaction 52 is finished, the log will not have the attestation for transaction 51 signed by the provider, User2 and the broker.

In this context, the blockade of simultaneous access ensures the order of the transactions. On the other hand, there is no possibility of violations resulting from the access to different files. In SCUDO, the concurrent access to different files is allowed, since a different chain of attestations is maintained for each file.

4.4 Auditing

The monitoring aforementioned is based on the management of transactions by an honest broker that enables the real-time detection of violations resulting from data loss or corruption. Due to the possibility of the broker also being malicious, SCUDO includes an auditing mechanism to identify the violations not previously detected. If no violation is reported in the auditing, it is demonstrated the security of the storage service, ensuring the security properties.

The data corruption is always detected in monitoring, because the file signature differs from that indicated in the attestation. Although a malicious broker cannot send a valid attestation to bypass this violation detection, this broker can:

- send an old attestation and metadata (with matching keys) in which the **FileVersion** and the **FileHash** are compatible with the file stored by the provider. This occurs when, for example, the provider loses the current version of a file and restore to a previous one. Thus, it is possible to receive and accept an outdated file, because the attestation wrongly indicates that the file is up-to-date, bypassing the violation detection in real-time.
- collude with a revoked user, enabling the creation of an attestation of an illegal reading/writing. The illegal writing is not detected in the next transaction, because this attestation is valid and reports a **FileHash** equal to the signature of the received file. In addition, an illegal reading cannot be detected before an auditing, because there is no modification of the stored file.

It is important to highlight the impossibility of data leakage from unauthorized users, because they cannot extract the encryption key from metadata. Thus, the revoked users can only read old versions of a file.

The auditing detects these violation scenarios. Before the auditing, the provider must send all attestations to TPA, and the broker and users send their last attestations. For each file, the TPA builds the chain of attestations, ordering and analyzing them. The cloud customer also sends the current permissions so that the analysis starts with the last attestation and ends with the first one. Algorithm 1 describes the auditing process of a single file, being an adapted version of the implementation adopted in this thesis. First, the process is initialized (lines 2-9) based on the received information to:

- configure the permissions;
- extract the Last Sequence Number (LSN) of each user;

- verify if the last attestation from the provider and broker are equals; and
- obtain the file version of the last attestation and the number of attestations.

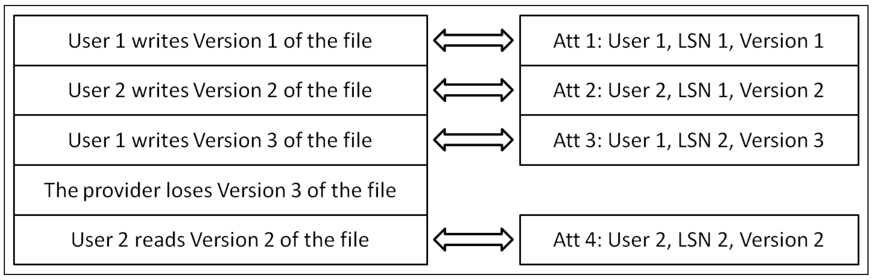
Next, each attestation of the chain is analyzed to prove that no violation occurs or report them (lines 10-32). Then, besides the correctness of the chain of attestations (lines 28-31), TPA checks the signature of the attestation (line 12), the user's permissions (line 16), and if LSN and the file version are the expected (lines 18 and 22). In addition, LSN is decremented (line 21), and the permissions and the expected version are modified in accordance with the type of transaction (lines 24-27). This verification is necessary to identify all attempts to deceive the violation detection.

Algorithm 1 – Auditing process

```
1  def audit():
2      perms = getPermissions()
3      LSNs = getLastUsersLSN()
4      lastBrokerAtt = getLastBrokerAtt()
5      lastProviderAtt = getLastProviderAtt()
6      if (not lastBrokerAtt.__eq__(lastProviderAtt)):
7          print("Error: Last atts differences")
8      expectedVersion = lastBrokerAtt.getFileVersion()
9      n = getNumberAtts()
10     while (n > 0):
11         att = getAtt(n)
12         if (not att.verifyAtt()):
13             print(Signature Error)
14         user = att.getUser()
15         lsn = att.getLSN
16         if (perms.hasPermissions(user, att)):
17             print("Violation: Transaction not Authorized")
18         if (lsn != LSNs[user]):
19             print("Violation: LSN absent")
20         else:
21             LSNs[user] -= 1
22         if (expectedVersion != att.getVersion()):
23             print("Freshness/Write-serializability Violation")
24         if (att.getTransType() != READING):
25             expectedVersion -= 1
26         if (att.getTransType() == UP_PERM):
27             perms.updatePermissions(att)
28         if (n > 1):
29             prevAtt = getAtt(n-1)
30             if (not att.verifyChain(prevAtt))
31                 print("Violation: Atts not chained")
32         n -= 1
```

Figure 15 describes an example scenario in which the provider lost the third version of a file and colluded with the broker to bypass the violation detection. It is clear that the sequence of versions does not follow the correct order so that the broker must send the second attestation (*Att2*) to deceive the real-time detection. Similarly, the provider chains *Att4* with *Att2* (and not *Att3*). Therefore, if the provider sends all attestations, the TPA detects a failure when sorting the attestations. As an alternative, the provider can try to deceive the detection, not sending the third attestation for auditing. However, *LSN2* of the *User1* will be missing due to the sending of the attestations by users, making this sending essential to verify if the provider hid some transaction.

Figure 15 – Example of Violation Scenario



User 1 writes Version 1 of the file	Att 1: User 1, LSN 1, Version 1
User 2 writes Version 2 of the file	Att 2: User 2, LSN 1, Version 2
User 1 writes Version 3 of the file	Att 3: User 1, LSN 2, Version 3
The provider loses Version 3 of the file	
User 2 reads Version 2 of the file	Att 4: User 2, LSN 2, Version 2

Source – The author.

On the other hand, sometimes, a user can detect the violation based on his/her last attestation. Based on the previous example, if *User2* performs the third transaction, he/she knows, in the next transaction, that the file version is, at least, three. Thus, this user can detect a violation when reading the second version of the file. This detection is not possible if the knowledge of the file version is lost. In this context, the users must keep one last attestation per file so that the attestation about one file is not overwritten after a transaction with another file.

The use of a single **UserLSN** and one chain of attestations for all files is enough to verify security properties. However, the use of one **UserLSN** and chain of attestations per file enable the auditing of different files in parallel. In addition, it is possible to probabilistically choose the files to be audited as suggested by Popa *et al.* (2011).

Due to the signature of attestations, no entity can deny a violation. After the auditing, the attestations can be discarded, except the last attestation of each file that will be chained with the attestations of the new epoch[2].

[2] The period between two consecutive audits is called **epoch** (POPA *et al.*, 2011)

4.5 Summary

This chapter described the security mechanisms used in SCUDO to protect the file stored in the cloud. The access control and monitoring/auditing are used to enable, respectively, file sharing and the verification of security properties. Thus, these properties are ensured and the trust of the storage services is improved. In addition, the limitations of existing solutions are overcome, dealing with scenarios of violations not detected by them.

Moreover, the use of an honest broker enables users to trust that the attestation and the metadata received are the last ones, making possible the real-time detection of violations. However, if the broker is compromised, the users may not detect the violations, because they do not have the correct information about the current state of the system. So, this makes the auditing indispensable to identify the undetected violations analyzing the historical transactions.

5 SCUDO EVALUATION

This chapter reports an evaluation of the security and performance of the SCUDO. The security analysis demonstrates the detection of violations resulting from the attacks specified in the threat model mentioned in Section 2.3 and is based on a formal model using Colored Petri Nets (CPNs) described in Section 5.1. In addition, the developed prototype is presented in Section 5.2, followed by the experiments performed in a real cloud environment in Section 5.3.

5.1 Security Analysis using CPNs

Formal methods have been used to evaluate security protocols, demonstrating their protection against existing threats, as discussed in Section 2.4. In this section, the modeling of the SCUDO shows its capacity to verify security properties.

The modeled CPN includes the protocols for auditing and for monitoring cloud transactions (*i.e.*, reading, writing and permissions updating) to analyze the detection of violations. This analysis demonstrates the robustness of the violation detection against existing attacks (see Section 2.3) and flaws found in the related work (see Chapter 3).

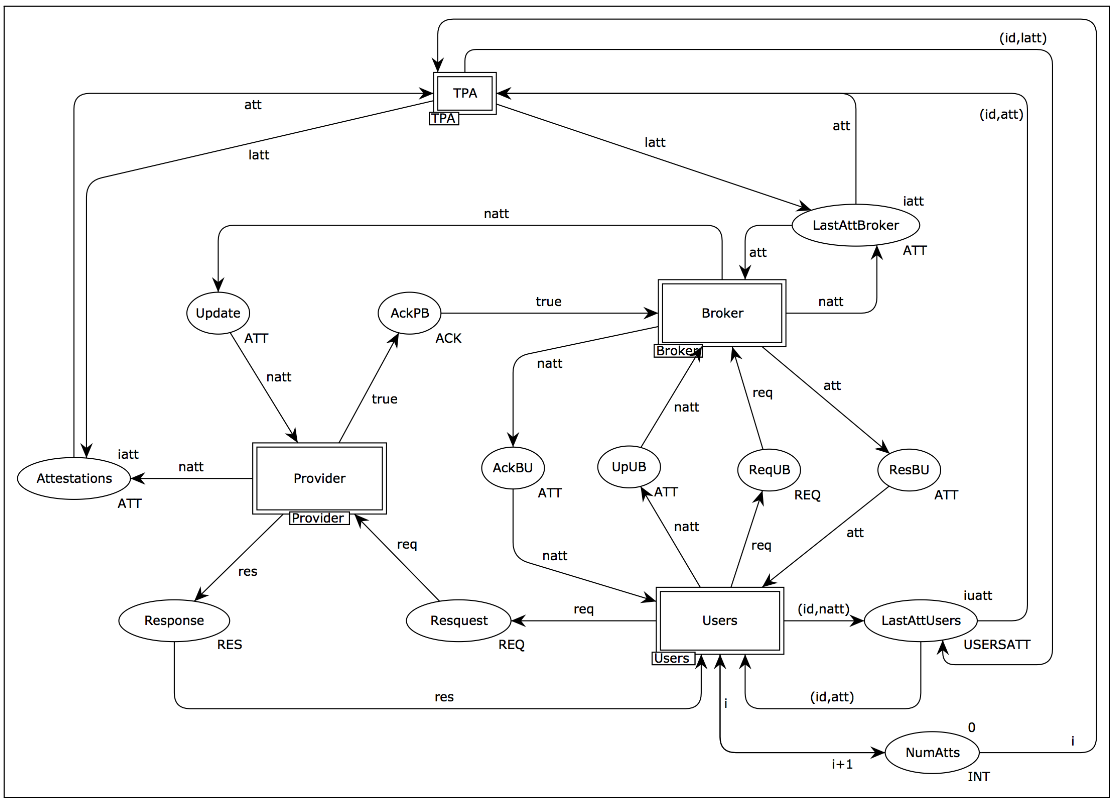
5.1.1 Overview

Figure 16 presents the overview of SCUDO modeling using CPN[1]. This model highlights the messages exchanged during the execution of each transaction and the sending of the attestations for auditing. The model of each sub-module, referring to each stakeholder, describes the state of the system so that the messages are exchanged in accordance with the protocols detailed in Chapter 4. Appendix A contains the images exported by the CPN Tools of the models of these sub-modules.

Simplifications are necessary to facilitate the analysis of the model, reducing its complexity, and they include: i) the storage of a single file; ii) the representation of the **ChainHash** as a sequence number; and iii) the assumption that all messages are authentic and that the file is encrypted. The modeled CPN is parameterized 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. It is important to mention that these simplifications do not change the SCUDO functioning.

[1] The original .cpn file of the CPN can be used for simulations and is available at https://goo.gl/265BdG.

Figure 16 – Overview of the simplified model of the SCUDO



Source – The author.

This model represents the correct behavior of the involved entities and simulations are realized to verify the correctness of the modeled mechanism. With these simulations, it is possible to observe the execution of transactions as specified in Chapter 4. At this moment, no 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 and shows to cloud users that the entities are trustworthy.

In order to validate the proposed mechanisms, it is necessary to include malicious actions in the model as recommended by Seifi (2014). In this context, the malicious behaviors of the provider and broker were modeled in different CPNs, demonstrating that the attacks are detected. The specification of two users in the CPNs was enough to detect the violations. An attack place is defined to indicate the execution of malicious actions by the provider and/or broker. Thus, every time that an attack is triggered, a token is sent to this place, denoting the necessity to detect a violation in the future. In this context, if a token is stored in this place, the monitoring or the auditing must also store a token in the error place, reporting the detection of a violation.

Table 4 presents the results of the validation, reporting when the violations are detected, in accordance with the attacks enumerated in Subsection 2.3. These are the attacks found in the literature related to the violations of freshness, write-serializability, confidentiality and integrity. Although users can also identify collusion attacks under special conditions, as described in Section 4.4, this table informs only the worst case of violation detection, indicating the detection of collusion attacks only in auditing time.

The remainder of this section details the detection of violations resulting from the attacks mentioned in Table 4. Videos of the simulations that demonstrate the violation detection of attacks are available at https://sites.google.com/site/candrebc/scudo.

Table 4 – Violation detection according to the malicious actions

Broker	Provider	Detection time
Honest	Send an outdated file	Monitoring
Honest	Write files out of order	Monitoring
Honest	Perform an unauthorized writing	Monitoring
Send an old attestation/metadata	Send an outdated file	Auditing
Send an old attestation/metadata	Write files out of order	Auditing
Send an old attestation/metadata	Perform an unauthorized writing	Auditing
Send an old attestation/metadata	Perform an unauthorized reading	Auditing
Send an old attestation/metadata	Honest	Monitoring

5.1.2 *Data Loss*

Data loss results in freshness or write-serializability violation when the provider restores the system to a previous state without reporting to the cloud customer. In this scenario, a malicious provider can execute a rollback attack and then send an outdated file during a reading transaction (see Section 2.3). In SCUDO, the broker sends the last attestation, informing the current version of a file to enable the detection, by the users, of freshness violations. If the broker is honest, this attestation is really the last one, so that the violation is detected in real-time, avoiding the use of old data.

Data loss results in a write-serializability violation if, after the rollback attack, a writing transaction is performed so that the new version overwrites a much older one. The writing out of order is avoided by SCUDO, recovering the stored file to check its version. The simulations show that this verification is essential for detection of the attacks, fixing flaws found in related work (POPA *et al.*, 2011) (CARVALHO *et al.*, 2016a) and being an important improvement of the violation detection mechanism.

In the model, the provider stores all versions of the file so that the attacks, that represent the data loss, are triggered when an old version is sent. For simplicity, a file is represented by its content and version, making it unnecessary to check the file hash. Thus, it is enough to compare the received version with the expected version sent by the broker, since all exchanged messages are authentic.

On the other hand, the broker can also be malicious and collude with the provider to bypass the violation detection. In this case, the synchronization between them is necessary so that the broker reports to users the same version of the file stored by the provider. In order to make this attack viable, the broker must also roll back to a previous state of the system. In the CPN modeling, the broker stores all attestations and metadata, so that when the attack is triggered, it sends the attestation and metadata in accordance with the file version indicated by the provider.

Due to the storage of the last attestation by the users, it is possible to detect the violation comparing the version registered in this attestation with the version displayed in the attestation received from the broker. However, the user cannot always discover if the received attestation is old so that the auditing is mandatory to identify the collusion attacks (see Section 2.3).

The auditing is performed as described in Section 4.4, building the chain of attestations and checking if the **UserLSN** and the **FileVersion** of each attestation are as expected. The analysis of the CPNs reveals that the collusion attacks are characterized by the absence of some **UserLSN**. In an attack, the provider and the broker rollback to a previous state of the system, removing the evidences (*i.e.*, the attestations) of the execution of the receding transactions. Thus, the absence of the **UserLSNs** of the removed transactions is identified, in auditing, based on the last attestations sent by the users. Otherwise, if all the attestations are sent, it would not be possible to validate the chain of attestations as well as there would be a failure in the order of the file versions. The detection of collusion attacks is not addressed by all related work (see Section 3) and is another improvement of this thesis.

In addition, SCUDO can identify if only the broker is malicious when informing an old attestation. In this case, the honest provider informs users of a different state and can send, for example, a newer version of the requested file. A user detects an inconsistency, because he/she cannot chain the attestation received from the provider with the attestation sent by the broker. The last attestation maintained by the provider is also signed by the broker and is used to prove, in auditing, the failure of the broker. The auditing also demonstrates whether only the provider is honest or the provider and the broker rollback to different states.

5.1.3 Data Leakage

The execution of an unauthorized reading results in data leakage while an unauthorized writing implies data corruption, affecting the data confidentiality and integrity, respectively. It is essential to highlight that the use of cryptography solutions is enough to detect or avoid illegal transactions from external entities because the keys are not available only for unauthorized entities (BONEH *et al.*, 2005) (JIN *et al.*, 2018). On the other hand, (KALLAHALLA *et al.*, 2003) reveal that revoked users can perform an illegal transaction in collusion with the provider. However, no related work discusses this issue (see Section 3). Then, this subsection focuses on the analysis of the data leakage, and the data corruption is discussed in the next section.

In the modeling, the **UserIDs** of the authorized users are indicated in the metadata so that a user can read a file only if his/her ID appears in the metadata sent by the broker. In this context, an old metadata can point out that a revoked user has permission to access a file, requiring the detection of attacks performed by revoked users. If only the provider is malicious, a revoked user cannot read the file, given the impossibility to extract the keys from the current metadata sent by the broker. Then, the data leakage is possible only if the broker is also malicious. For this reason, Table 4 shows only the detection of unauthorized writings in monitoring.

Although a revoked user cannot decipher the current metadata, the broker can send a previous one, making he/she able to access a file or sign a new file. Revoked users can only read old versions of a file because of the immediate re-encryption when updating permissions. By dealing with different file versions, the **FileHash** of the last attestation is different from the signature of the received file. Thus, the illegal reading is performed with success if: i) the revoked user ignores the verification of integrity; or ii) the broker sends an old attestation.

If this unauthorized reading is included in the chain of attestations, a user will detect the violation in the next transaction, when accessing the current version of the file and the last attestation report a previous one. In order to deceive the violation detection, the malicious provider and the broker can chain the unauthorized reading with an old attestation, using the old file from now on. This case is similar to the rollback attacks, receding some transactions. Thus, the auditing will report a violation due to the absence of **UserLSNs** of the removed transactions.

When analyzing the impact of the lazy revocation approach, it was possible to observe that a revoked user can read a file if it has not been re-encrypted yet. The use of lazy revocation enables that the attestation of this reading be chained normally with the last attestation so that the violation is not detected in real-time. In this case, the violation is detected when checking the user's permissions during the auditing. However, it is important to mention that only the broker is responsible for checking the permissions so that an honest provider can normally send the current version of the file. Thus, in this scenario, it is possible to avoid data leakage if the lazy revocation is not used. Therefore, the use of this approach is not a security recommendation, since it is better to avoid an attack than to detect it.

On the other hand, the experiments indicated SCUDO detects unauthorized readings only if all transactions were registered in the log and sent for auditing. Thus, the violation detection is bypassed if the malicious provider and the broker ignore this reading, not including it in the chain of attestations. In order to ensure the detection of data leakage, solutions of trusting computing must be used so that accountability tools register all events in logs that are tampered-free and cannot be deleted (KO *et al.*, 2011). However, the study of these tools is beyond the scope of this work.

5.1.4 Data Corruption

Unlike the scenario of data leakage, it is not possible to deceive the violation detection without registering an unauthorized writing in the log. The detection of integrity violations in real-time is ensured by the verification of the digital signatures, avoiding the use of corrupted data. Similarly, it is possible to ensure the update of permissions only by the cloud customer, due to the signature of the file metadata.

The real-time detection relies on an honest broker, and this detection is based on the comparison between the signature of the received file and the one indicated in the attestation sent by the honest broker. In this context, unauthorized users cannot write files without the violation being detected. This violation scenario is properly addressed by related work (see Chapter 3) so that this analysis focuses on the writings performed by revoked users that use a previous key to write the file. In this case, the attack is not detected in real-time only with the malicious collaboration of the broker because it is not possible to have an attestation without the signature of the broker. Thus, a collusion attack is required so that the malicious broker signs an attestation, informing that the signature of the file is valid even if the key is old.

It is essential to mention that the file metadata is also sent to users by the broker. The metadata is represented in the CPNs by the list of users with their permissions and the key version. Then, a malicious broker sends an old metadata, indicating that the revoked user has permission to write the file. If the provider is honest, it reports a violation comparing the key version used to sign the file with the version indicated in its last attestation. Then, although SCUDO does not use the lazy revocation, it is important to keep the field **KeyVersion** in the metadata and the attestation.

In a collusion attack, an unauthorized writing generates an attestation signed by the provider, the broker and the revoked user. In the next transaction, the broker must send this attestation and the same metadata used by a revoked user for writing to inform the correct key for the verification of integrity. Although the **KeyVersion** in the attestation and the metadata is the same, the user can report the violation based on the **KeyVersion** of his/her last attestation.

However, it is not possible to ensure that the users hold an attestation in which the **KeyVersion** is bigger than the one reported by the broker. Due to the impossibility to ensure the detection of an unauthorized writing before a new writing, the auditing is mandatory to identify all scenarios of collusion attacks by checking the permissions of each transaction.

5.2 Implementation and Deployment of SCUDO Prototype

A prototype of SCUDO was developed using Python 3.6 and deployed in the cloud infrastructure of GREat (Group of Computer Networks, Software Engineering and Systems) at UFC (Federal University of Ceará).

Figure 17 shows aspects of the implementation of SCUDO and its deployment in a cloud infrastructure based on OpenStack[2]. In a nutshell, the servers are two instances with Linux Ubuntu 14.04.5 LTS (Long Term Support) that execute the broker and the provider applications. When these applications are executed, they wait for connections from clients that are closed after each transaction. The client machine is an HP EliteDesk computer running Windows Seven and is located at the Federal University of Piauí. The client application is executed, on the client, by a typical user or administrator user who requests the transactions by command line. Users can also request a list with the files stored in the cloud and only the administrator can write new files and update the permissions. In addition, for simplicity, the auditor also runs on the client machine, and the auditing data (*i.e.*, attestations and metadata) is manually downloaded from the cloud.

The prototype was developed in accordance with the specification (see Chapter 4), and its source code contains about 1,700 lines and is available at https://goo.gl/265BdG. Although a single computer is used in the experiments, the client application can be executed from different computers. Besides, the sending and receiving of files during the transactions, the

[2] https://www.openstack.org/

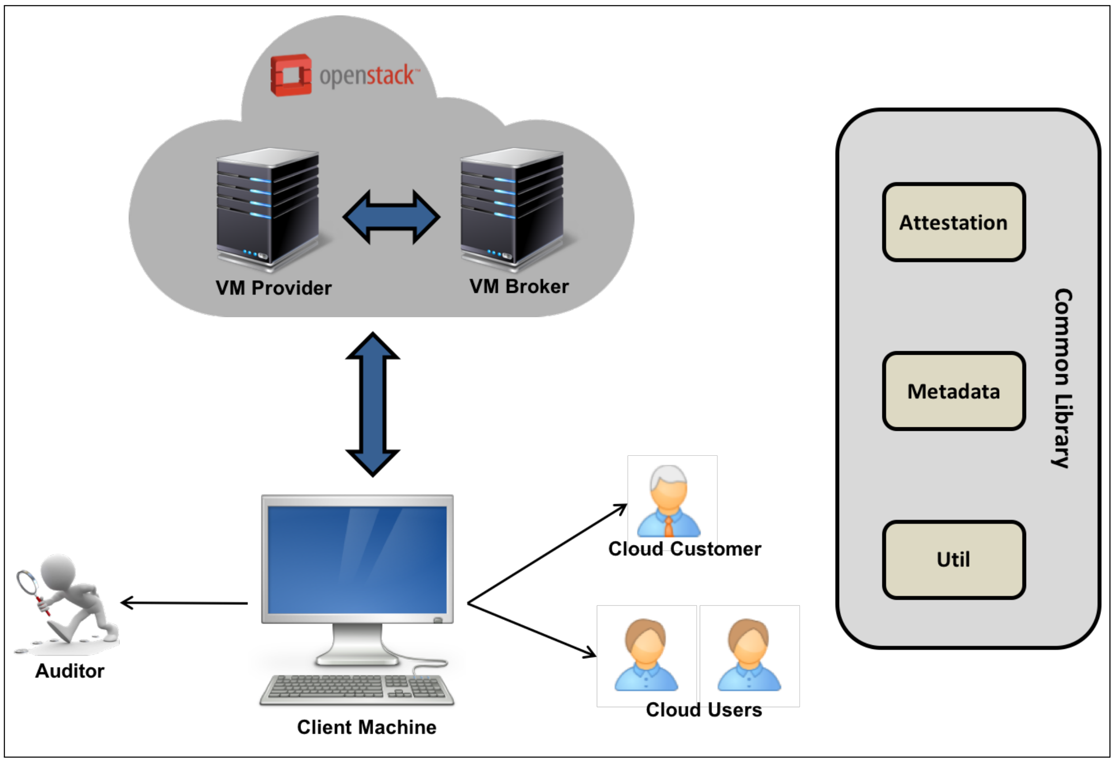
users keep a copy of the last attestation of each file for auditing purposes.

The provider, the broker and the users have digital certificates that are used for network communication with SSL/TLS and for the signatures of the attestations and metadata. Figure 18 shows an example of an XML file of an attestation. The metadata is also stored as an XML file, indicating the permissions and the keys. The common library contains functions to manipulate these files and is used by the stakeholders during the transactions to access the information stored in the metadata and the attestations. This library also contains a package with auxiliary functions for network communication and cryptography. It is worth pointing out that this implementation is used as the proof of concept for the experiments so that more efficient implementations can be developed.

5.3 Experiments

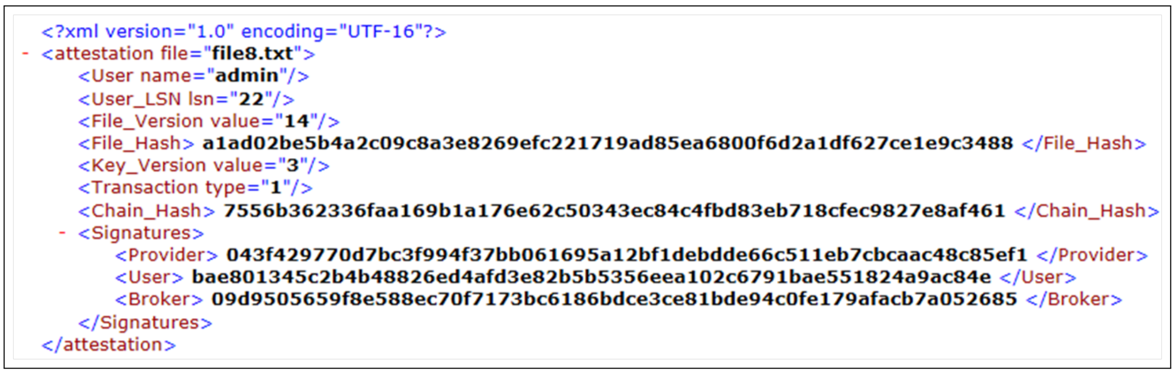
Experiments with the SCUDO prototype were performed to show the detection of violations and the performance of the monitoring and auditing mechanisms. These experiments and their results are described in the next sections.

Figure 17 – SCUDO Implementation and Deployment



Source – The author.

Figure 18 – Example of an attestation



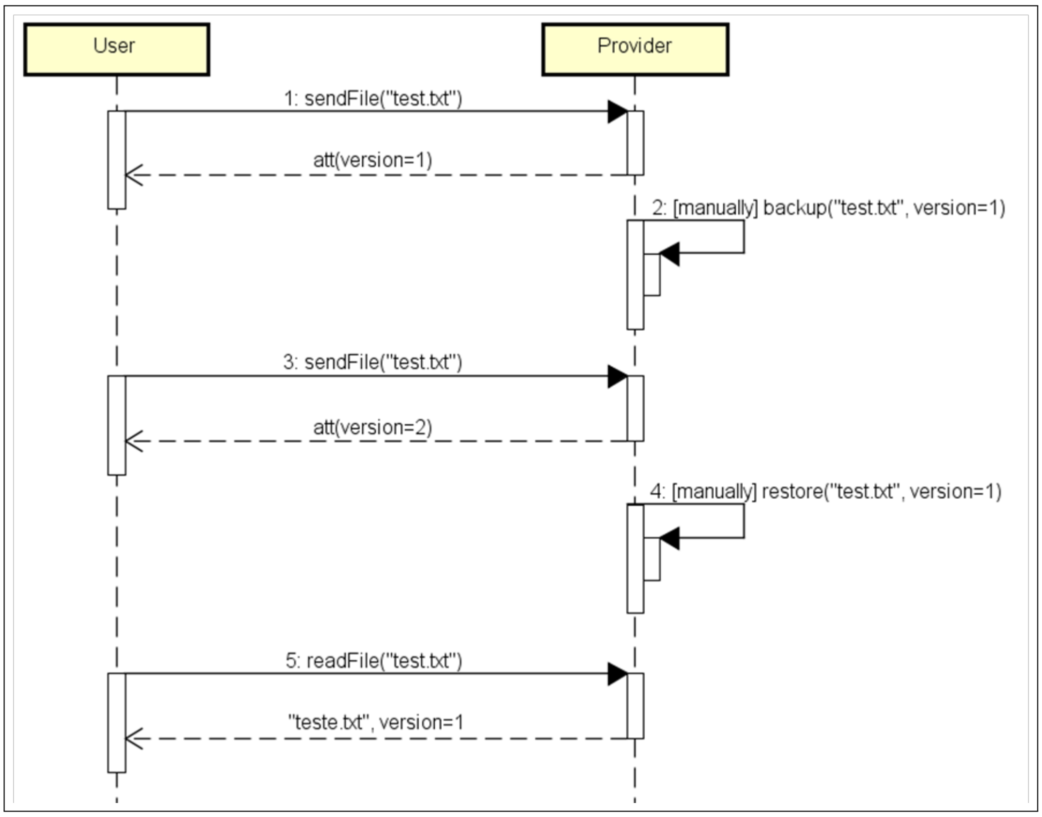
```
  <?xml version="1.0" encoding="UTF-16"?>
- <attestation file="file8.txt">
      <User name="admin"/>
      <User_LSN lsn="22"/>
      <File_Version value="14"/>
      <File_Hash> a1ad02be5b4a2c09c8a3e8269efc221719ad85ea6800f6d2a1df627ce1e9c3488 </File_Hash>
      <Key_Version value="3"/>
      <Transaction type="1"/>
      <Chain_Hash> 7556b362336faa169b1a176e62c50343ec84c4fbd83eb718cfec9827e8af461 </Chain_Hash>
    - <Signatures>
          <Provider> 043f429770d7bc3f994f37bb061695a12bf1debdde66c511eb7cbcaac48c85ef1 </Provider>
          <User> bae801345c2b4b48826ed4afd3e82b5b5356eea102c6791bae551824a9ac84e </User>
          <Broker> 09d9505659f8e588ec70f7173bc6186bdce3ce81bde94c0fe179afacb7a052685 </Broker>
      </Signatures>
  </attestation>
```

Source – The author.

5.3.1 *Detection of violations*

The developed prototype implements the correct behavior of the stakeholders so that the attacks were simulated by modifying manually the files stored in the provider and/or the broker. Figure 19 describes the Sequence Diagram of an attack in which the current file stored is overwritten by an old version to indicate data loss. For simplicity, the broker does not appear in this figure since there is no change in its behavior.

Figure 19 – Example of an attack (data loss)



Source – The author.

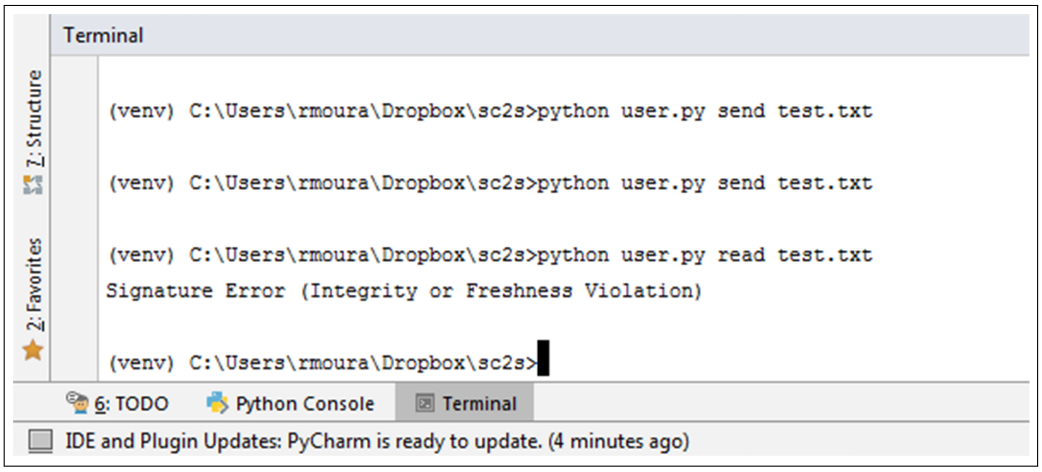
The execution of the attack depicted in Figure 19 results in a violation detected when reading the file, as shown in Figure 20. The signature error is reported because the file hash indicated in the attestation sent by the broker differs from the signature of the received file. This error can denote a freshness violation, as in the described attack, or an integrity violation, stating that the file is corrupted. Thus, data corruption is detected similarly.

In order to analyze the collusion attack, in addition to the file backup, the provider and the broker make a backup of the first attestation. After the second transaction, they replace the last attestation by the backup so that the third attestation is chained to the first one. Other users will not detect the violation since they do not know about the second transaction. In this case, if the provider sent, for auditing, all attestations, the violation is reported, because the attestations were not well chained. Otherwise, due to the lack of the attestation of the second transaction, the user's sequence number of this transaction is absent (see Figure 21).

The revoked users can perform illegal transactions colluding with the provider and broker. In this case, no violation is detected in monitoring, but the auditing will report it. The simulation of this attack is performed, before revoking the permissions, the old metadata is stored together with the attestation and the file. Thus, after the administrator updates the permissions, this information is restored, so that the revoked user can extract the keys and write a file. This attestation is normally chained with the following attestations, trying to deceive the real-time detection. In this case, the auditing reports the illegal transaction based on the metadata sent by the administrator.

Videos executing experiments are available at https://goo.gl/265BdG, and the detailed

Figure 20 – Commands executed by the user during this experiment

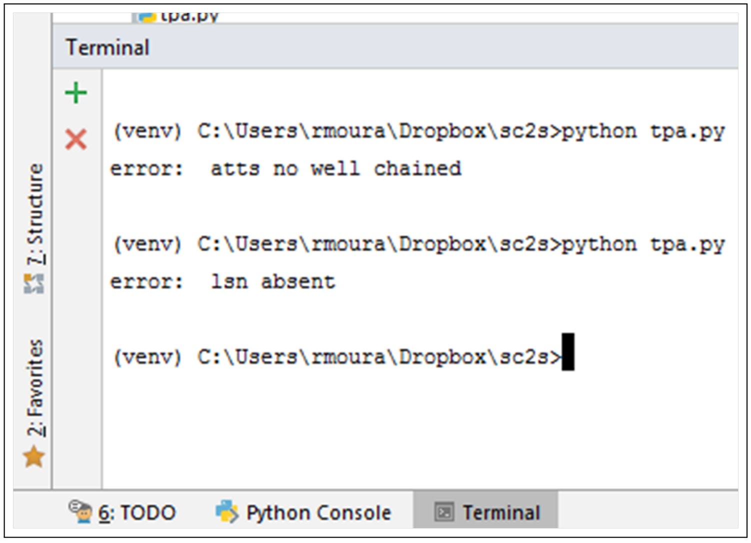


```
(venv) C:\Users\rmoura\Dropbox\sc2s>python user.py send test.txt

(venv) C:\Users\rmoura\Dropbox\sc2s>python user.py send test.txt

(venv) C:\Users\rmoura\Dropbox\sc2s>python user.py read test.txt
Signature Error (Integrity or Freshness Violation)

(venv) C:\Users\rmoura\Dropbox\sc2s>
```

IDE and Plugin Updates: PyCharm is ready to update. (4 minutes ago)

Source – The author.

Figure 21 – Detection of data loss during the auditing



```
(venv) C:\Users\rmoura\Dropbox\sc2s>python tpa.py
error:  atts no well chained

(venv) C:\Users\rmoura\Dropbox\sc2s>python tpa.py
error:  lsn absent

(venv) C:\Users\rmoura\Dropbox\sc2s>
```

Source – The author.

security analysis was described in Section 5.1.

5.3.2 Performance of the execution of transactions

The protocols to execute cloud transactions add an overhead related to the cryptographic algorithms and additional messages with the attestations and metadata. So, experiments were performed in this work to calculate this overhead and analyze the impact when using different files sizes (8KB, 32KB, 128KB, 512KB and 1024KB). For each file size, the transactions were executed thirty times, extracting their average and the standard deviation[3]. Appendix B presents the complete results, in which it is possible to observe that the standard deviation times are shorter when compared to the average times, showing homogeneous results.

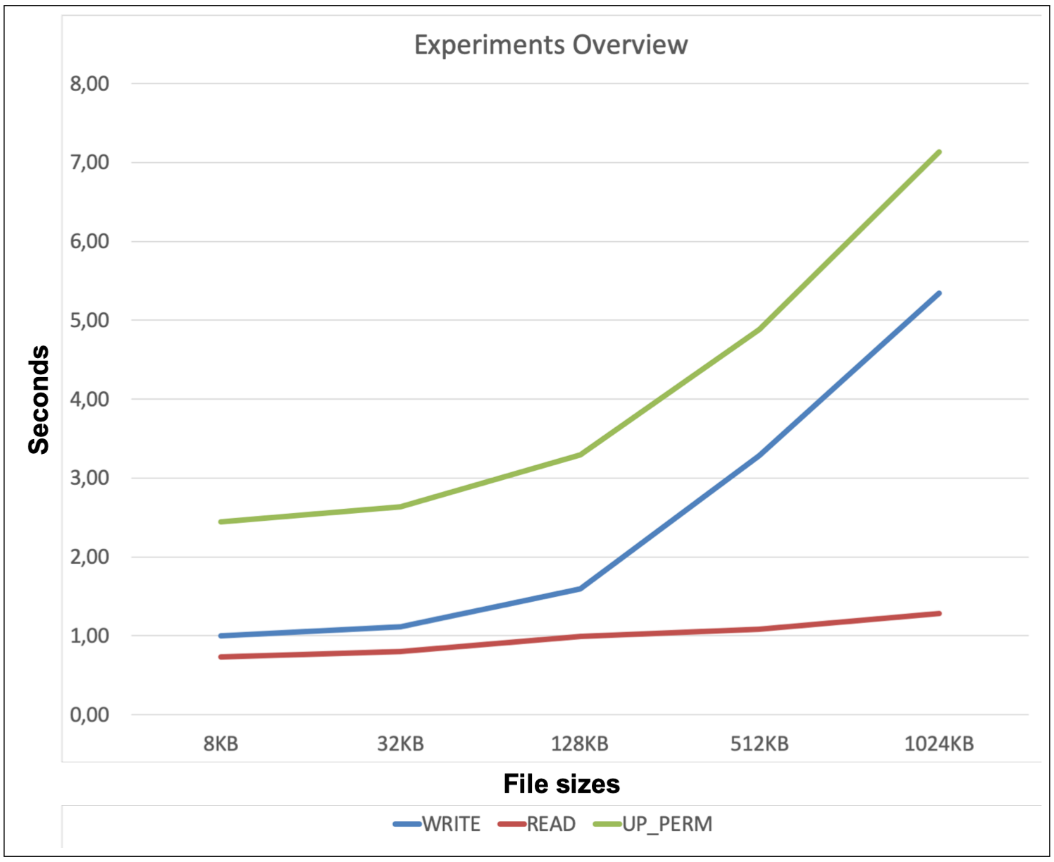
The average time, in seconds, of the transactions of reading, writing and updating of permissions are displayed in Figure 22. As shown in the figure, the execution time increases when using the larger files given the time needed to encrypt/decrypt and send/receive these files. Due to the higher speed for download files, the reading transactions have better results. Even so, the execution time increases linearly with the size of the file, regardless of the type of transaction. The growth in transactions for writing and permission updating seems to be exponential, but this is expected due to an exponential increase in file size. Times for reading transactions are reduced due to high download speed.

[3] Thirty repetitions are commonly performed in the experiments for statistical analysis (WEI *et al.*, 2007) (LOO, 2005) (BROBERG *et al.*, 2009)

Figure 23 details the cost related to overhead, cryptography and download of reading transactions, in accordance with the size of the files. The overhead cost is related to the exchanged messages involving the metadata and attestation, specified in the protocol to read a file (see Section 4.3.2). These messages are unaffected by the files sizes and sending them takes constant time (on average, 0.49 seconds). In addition, it is possible to observe the low cryptographic cost (from 0.02s to 0.14s) when comparing with the cost to download the files. This behavior is similar when writing files, and the difference is the greatest impact of the upload time, ranging from 0.11 to 4.45 seconds, according to the size of the file. On the other hand, the difference in download time is smaller, varying from 0.2 to 0.66 seconds in the experiments.

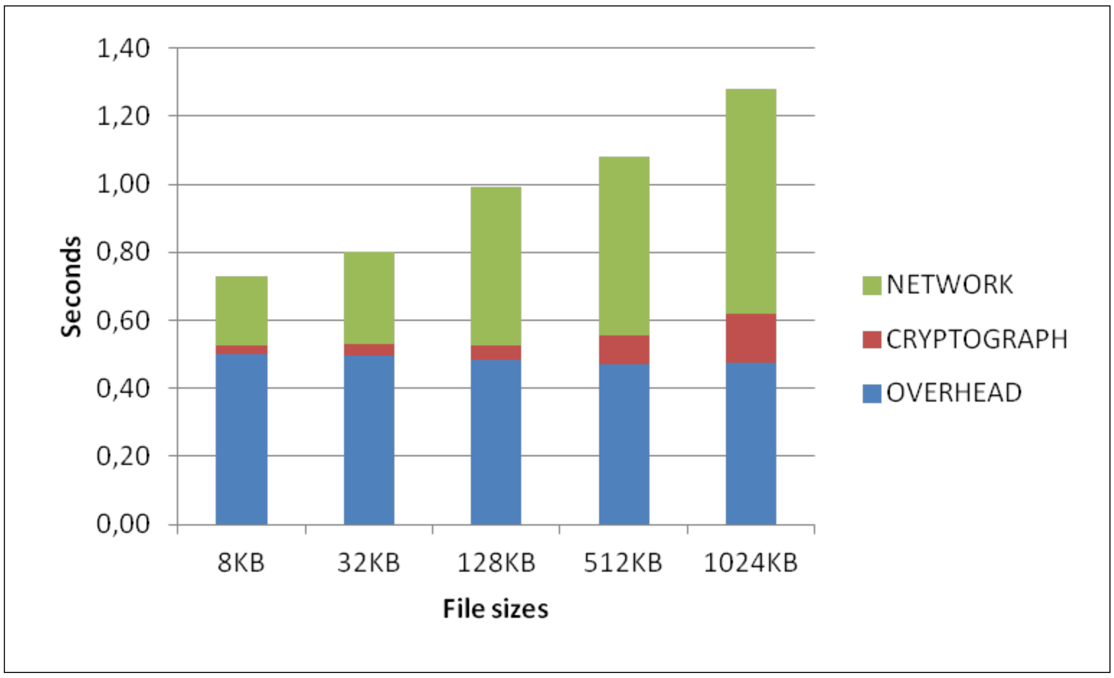
The impact of the upload time is also visible when observing the transactions for permissions updating. Thus, the files are downloaded from the provider, re-encrypted with new keys and sent back. Due to of the loss of security when using the lazy revocation, discussed in Section 5.1.3, the lazy revocation was not developed in this prototype, resulting in a higher cost related to the re-encryption with the new keys. Figure 24 shows, besides the overhead, the cost to re-encrypt and to generate/distribute the new keys. In this figure, the time to re-encryption

Figure 22 – Transactions execution time



Source – The author.

Figure 23 – Reading transactions



Source – The author.

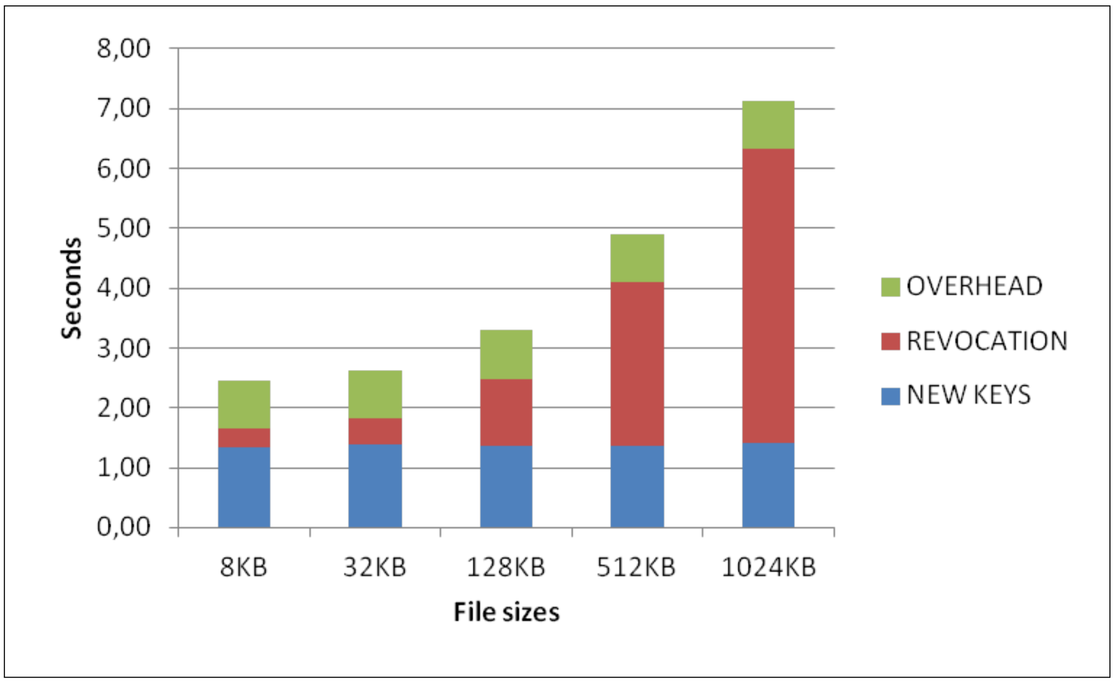
goes from 0.32s to 4.91s, including the time to download and upload the file. The analysis of the transactions of reading and writing shows that the network cost (upload and download) is higher than the cryptography cost. The cost to generate/distribute the new keys does not depend on the size of the files, and is, on average, 1.37 seconds. It is interesting to mention that 1.07 seconds is used to create the keys while the rest is for their distribution.

5.3.3 Performance of auditing

A TPA (Third-Party Auditor) performs audits to detect the collusion attacks or demonstrate the honesty of the stakeholders. Figure 25 displays the auditing time in accordance with the number of verified attestations. For each log size, thirty repetitions were performed, extracting their average. Although the verification of an attestation has taken, on average, 0.1 seconds, it is possible to improve the auditing time using parallel or distributed computing in which each processor or computer checks a subset of the attestations.

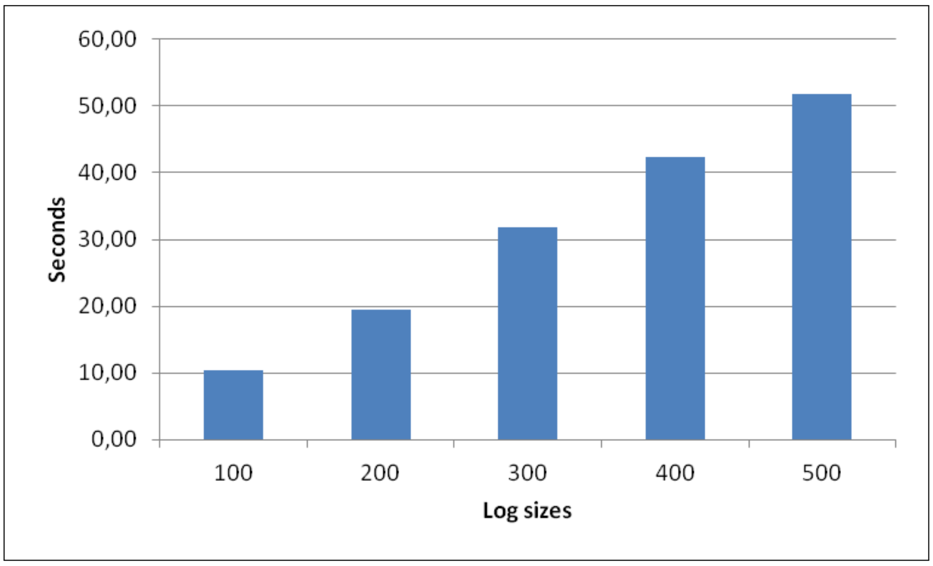
It is important to mention that it was not possible to compare these results with the related work due to the differences between the experimental environments. However, it is possible to observe that the time to execute the transactions grows linearly with the file size, just as the auditing time depends linearly on the size of the log. In addition, the cost of communication has a greater impact than the cost of security when increasing the size of the

Figure 24 – Permissions updating transactions



Source – The author.

Figure 25 – Average time of auditing



Source – The author.

files. Therefore, this analysis indicates an adequate performance of SCUDO.

5.4 Summary

This chapter presented the results of the evaluation of the SCUDO based on its CPN model and prototype. The model was used for a thorough security analysis. The identified attacks were included in the modeling and detected by monitoring or auditing. Different scenarios of attacks were analyzed, also analyzing the attempts to deceive the violation detection. These attacks were detected as expected. In addition, the use of a trust computing solution was helpful to detect data leakage.

The analysis of the prototype indicated adequate performance. In addition, experiments were performed to exhibit scenarios of violation detection. In this context, the security mechanisms at SCUDO allowed the provider and the broker to ensure security properties, showing that they are honest.

6 CONCLUSION

This thesis describes a secure cloud storage service, called SCUDO, which focuses on the assurance of the following security mechanisms for data stored and shared in the cloud: confidentiality, integrity, freshness and write-serializability. These mechanisms protect the customers against data leakage and enable the violation detection, demonstrating the security of a storage service.

This chapter concludes this thesis, first, presenting in Section 6.1 its overview. The main results are described in Section 6.2. Then, the research hypothesis and questions are discussed in Section 6.3 as well as the limitations of this work (see Section 6.4). Finally, suggestions for future work are enumerated in Section 6.5.

6.1 Overview

The literature reveals solutions that address security issues in cloud storage, especially regarding the detection of data corruption, loss and leakage. In the cloud storage solutions, cryptographic algorithms and access control mechanisms have been used to avoid attacks, protecting the data stored in the cloud, and allowing data sharing (THILAKANATHAN *et al.*, 2014).

However, it is also necessary the use of monitoring and auditing mechanisms for detecting attacks that cannot be avoided (POPA *et al.*, 2011) (HWANG *et al.*, 2014) (JIN *et al.*, 2018). These attacks involve malicious actions performed by the stakeholders and result in violations of security properties. The monitoring and auditing mechanisms increase the transparency of cloud storage services, enabling the detection and proof of violations. So, it is possible to prove the honesty of the provider or its malicious behaviors using these mechanisms, which are the focus of this research.

As discussed in Chapter 3, existing solutions do not detect all security violations and fail in demonstrating their security. It is possible to highlight the following scenarios: i) the absence of detection of violations resulting from reading/writing transactions performed by revoked users; ii) the existence of scenarios in which the write-serializability violations are not reported; and iii) the necessity to detect the collusion attacks. In order to address these issues, SCUDO was proposed in this thesis.

In the first scenario, it is necessary to verify if each transaction was authorized. This verification is based on the transactions for permissions updating that are also stored in the log. Section 4.4 describes how this verification is performed by SCUDO.

For the second scenario, related to write-serializability violations, this research concluded that it is essential to verify the ownership of a file before each writing, as detailed in Section 4.3.2.

The violations are detected in real-time based on the information sent by an honest broker. In this research, it is also analyzed the possibility of a broker being malicious so that the collusion attacks are detected (third scenario). These attacks are detected in auditing as specified in Section 4.4.

In addition to the mechanisms for violation detection exemplified by these scenarios, this thesis also describes in Section 4.2 how access control is performed by SCUDO so that the proposed storage service detects violations of security properties in a data sharing environment, achieving the goal of this research.

Moreover, the security evaluation of the SCUDO was essential to show that the attacks specified in Section 2.3 are detected.

Last, the modeling with CPNs enabled the formal validation of SCUDO that proves the detection of the modeled attacks and the necessity of auditing to detect the collusion attacks and the reading by revoked users. Experiments with a prototype were also performed in this work to show the detection of attacks in a cloud infrastructure as well as for performance evaluation.

6.2 Main Results and Publications

All the results of this research were achieved during the activities defined in the methodology (see Section 1.5). After doing one or more activities, the partial results were submitted for publication as cited as follows:

- Analysis of existing solutions about security SLA for clouds and identification of research opportunities (CARVALHO *et al.*, 2017b);
- Identification of flaws in security evaluation of existing solutions with the use of formal methods (CARVALHO *et al.*, 2016a);
- Adaptation of the writing protocol defined by Popa *et al.* (2011) to properly detect data loss; and identification of violation scenarios resulting from collusion attacks, proposing their detection in auditing (CARVALHO *et al.*, 2017a);
- Specification of a mechanism to verify the correctness of the access control to detect

integrity violations, resulting from writing transactions performed by revoked users; and investigation of the impact of using the lazy revocation approach and scenarios of confidentiality violations that require the use of trust computing solutions for their detection (CARVALHO *et al.*, 2018).

It is also interesting to mention the publication of the initial proposal of this thesis in the Doctoral Symposium of the International Symposium on Cluster, Cloud and Grid Computing (CARVALHO *et al.*, 2017c). Although outside the scope of this research, two other papers were also published during the doctorate as results of collaboration with a master student (ANDRADE *et al.*, 2018) and with the industry in a project to detect vulnerabilities in Android operating systems (CARVALHO *et al.*, 2016b).

Table 5 enumerates all publications done during this doctorate.

Table 5 – List of publications

No.	Reference	Participation	Type
1	CARVALHO, C. A. B. de; ANDRADE, R. M. de C.; CASTRO, M. F. de; AGOULMINE, N. Modelagem e detecção de falhas em soluções para armazenamento seguro em nuvens usando Redes de Petri Coloridas: Um estudo de caso. In: **Workshop de Computação em Clouds e Aplicações (WCGA/SBRC)**, 2016. p. 17–30.	Main Author	Workshop
2	CARVALHO, C. A. B. de; ANDRADE, R. M. de C.; MAIA, M. E.; ALBUQUERQUE, D. M.; PEDROSA, E. T. O. Neutralizing vulnerabilities in android: a process and an experience report. **International Journal of Computer Science and Information Security**, v. 14, n. 3, p. 20-29, 2016.	Main Author	Journal
3	CARVALHO, C. A. B. de; ANDRADE, R. M. de C.; CASTRO, M. F. de; COUTINHO,E. F.; AGOULMINE, N. State of the art and challenges of security SLA for cloud computing. **Computers and Electrical Engineering**, v. 59, p. 141-152, 2017.	Main Author	Journal
4	CARVALHO, C. A. B. de; AGOULMINE, N.; CASTRO, M. F. de; ANDRADE, R. M. de C. How to improve monitoring and auditing security properties in cloud storage. In: **Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC)**, 2017. p. 559-572.	Main Author	Conference
5	CARVALHO, C. A. B. de; CASTRO, M. F. de; ANDRADE, R. M. de C. Secure cloud storage service for detection of security violations. In: **Doctoral Symposium of the International Symposium on Cluster, Cloud and Grid Computing (CCGrid)**, 2017. p. 715-718.	Main Author	Doctoral symposium
6	CARVALHO, C. A. B. de; ANDRADE, R. M. de C.; AGOULMINE, N.; CASTRO, M. F. de. Detection of access control violations in the secure sharing of cloud storage. In: **International Conference on Cloud Computing and Services Science (CLOSER)**, 2018. p. 124–135.	Main Author	Conference
7	ANDRADE, R. M. de C.; CORREIA, M. A. S.; CARVALHO, C. A. B. de; XIMENES, P. Evaluate Location Features for Continuous Authentication with Machine Learning Experiments. In: **International Workshop on ADVANCEs in ICT Infrastructures and Services (ADVANCE)**, 2018. p. 13–20.	Co-Author	Conference

6.3 Revisiting hypothesis and research questions

Section 1.3 presented the hypothesis and its respective research questions that guided this work: *it is possible to design and combine security mechanisms in a secure storage service to improve the violation detection of confidentiality, integrity, freshness and write-serializability.* This hypothesis is **Accepted** since the proposed security mechanisms, designed and combined in Chapter 4, improve the detection of violations of security properties. This chapter also shows the combination of ACLs (Access Control Lists) and cryptographic algorithms to protect file sharing in the cloud. In addition, they are integrated with the designed mechanisms for violation detection.

These improvements are better visualized when comparing SCUDO with related work, as discussed in the answer to the second question and presented in Table 6. The discussion of the fifth research question is also important to demonstrate these improvements when validating SCUDO, showing the success in detecting violations of the chosen mechanisms.

In addition, the answer to the first research question presents the requirements of SCUDO, and issues related to the violation detection are shown in the analysis of the third and fourth questions. These requirements reveal important aspects of the design of this solution. For example, the attestations must be signed for non-repudiation purposes, and the keys must be distributed efficiently only for authorized users and modified when updating the permissions.

6.3.1 Research question 01

What are the requirements of secure cloud storage service to allow data sharing and the violation detection of the security properties?

Chapter 4 details the mechanisms for file sharing and violation detection. That chapter describes the requirements of each mechanism.

The access control mechanism requires schemes to distribute and manage keys and procedures for granting or revoking permissions. These schemes must prevent even the provider from accessing sensible information. The lazy revocation approach is an optional requirement that aims to improve efficiency when updating the permissions, but it is not a security recommendation.

The monitoring and auditing mechanisms are used to verify security properties, detecting potential violations. These mechanisms enable to prove if the cloud provider is trustworthy or if there have been violations. Then, the main requirements are the non-repudiation of a detected violation and the protection against false accusations.

The sending of the last attestation by the users and the broker allow identifying the provider hidden some transaction. On the other hand, a revoked user can read a file, and the violation detection is bypassed if the provider, the broker and the user do not report this transaction for auditing. So, a trust computing solution is required to avoid the removal of log entries and detect confidentiality violations. In this context, the designed mechanisms must consider the particularities of each existing attack that result in violations of the analyzed security properties.

6.3.2 Research question 02

What are the attacks that violate these properties and which ones are not detected by existing solutions?

Typically, a secure solution protects the stakeholders against external attackers. For example, network protocols include mechanisms to avoid attacks such as eavesdropping, replay attacks and modification of messages (STALLINGS, 2016). In this case, the parties are identified by an authentication protocol and are considered honest. However, in the cloud environment, it is recommended the protection of data sent to public providers, and the customers request the assurance of security properties.

In this context, it is fundamental to identify the malicious behaviors of a provider such as the execution of an unauthorized transaction, the sending of outdated data and the data loss. These attacks cannot be avoided, making essential the design of mechanisms to detect the resulting violations. Chapter 3 describes the attacks that are not addressed by the related work and Table 6 summarizes the improvements obtained with SCUDO in comparison with them.

Table 6 – Comparison of SCUDO with the related work

Work	Security Properties				Real-time Detection	Colusion Attacks
	I	F	W	C		
(POPA et al., 2011)	Partially	Yes	Partially	No	I	Unfeasible
(ALBESHRI et al., 2012)	Partially	Yes	Partially	No	I	Unfeasible
(HWANG et al., 2014)	Partially	Yes	Partially	No	I, F, W	Not detected
(TIWARI; GANGADHARAN, 2015a)	Partially	Yes	Partially	No	I	Unfeasible
(HWANG et al., 2016)	Partially	Yes	Partially	No	I	Unfeasible
(JIN et al., 2018)	Partially	Yes	No	No	I, F	Unfeasible
SCUDO	**Yes**	**Yes**	**Yes**	**Yes**	**I, F, W**	**Yes**

I - Integrity; F - Freshness; W - Write-serializability; C - Confidentiality

This table shows that integrity violations are partially identified by related work because it does not analyze the execution of transactions by revoked users. Similarly, the write-serializability violations are partially detected by related work because this detection focuses only on the log analysis, ignoring the content of the stored file.

The collusion attacks are inherent only to solutions that use a third party, here called broker, to manage the execution of the transactions. This third party is used by Hwang *et al.* (2014) to enable the real-time detection of violations. However, the collusion attacks are not analyzed by them because it is supposed that the broker is honest. On the other hand, in this research, it is analyzed the possibility that the broker is also malicious.

Last, Table 6 also indicates that confidentiality violations cannot be detected in real-time by SCUDO. This occurs because confidentiality violations are the result only of collusion attacks in which the provider and the broker send an old file and its reading key to a revoked user. In general, the collusion attacks are executed trying to bypass the real-time detection of violations, so that the detection of these attacks is only guaranteed in the auditing.

6.3.3 Research question 03

How can a storage service be monitored and audited to verify the security properties?

Chapter 4 describes the SCUDO and its monitoring and auditing mechanisms. These mechanisms detect the violations in accordance with the security property analyzed. The verification of integrity is performed in monitoring and based on the signatures of the files and messages. However, only this verification cannot be enough to detect a violation resulting from a writing performed by a revoked user. In this case, if the broker is also malicious, the violation is identified only in auditing when checking the permissions of each transaction.

In order to verify the freshness and write-serializability, the logs of transactions are analyzed. However, the violation of these properties can be detected by monitoring or auditing. In monitoring, the broker sends the current state of the system, allowing users to detect a violation briefly. If the broker is not honest, the logs of all transactions are ordered and analyzed, in auditing, to identify the violation. The logs must be signed for non-repudiation purposes. In general, it is necessary to analyze how an attack is performed and what information is used to identify the resulting violation.

The confidentiality is provided by an access control mechanism that distributes the keys only to authorized users. However, a collusion attack can allow the reading by a revoked user. In this case, the auditing detects a violation when using a trusting computing solution.

6.3.4 Research question 04

Are there attacks in which the resulted violations cannot be identified in real-time, requesting an auditing to detect these violations?

In order to answer this question, it is necessary to find at least one scenario in which the auditing is mandatory. It would be interesting that all violations could be detected as soon as possible, but, if the broker is not trustworthy, some violations' scenarios can be identified only in an auditing.

Section 4.4 describes an example in which the provider performs a collusion attack with the broker so that the broker does not report the loss of the current version of a file. The user that requested the writing of this version can detect a violation during his/her next transaction, but he/she can never perform another transaction (*e.g.*, when his/her access is revoked). The other users could not detect the violation, because the broker does not have the right information of the last transaction. Thus, it is not possible to ensure that a violation is always detected when receiving an outdated file so that the auditing is required to ensure this detection.

6.3.5 Research question 05

How can the security mechanisms be evaluated to demonstrate the security of the storage service, ensuring the non-violation of security properties or proving the occurrence of an attack?

The security evaluation must be based on the threat model of the analyzed mechanism to prove the protection against attacks. For example, in an evaluation of a cryptographic algorithm, it is verified the possibility of obtaining the secret key or the plaintext. Cryptanalysis' methods that must show that a symmetric cipher cannot be broken with less effort than the force brute. The scientific community frequently reports the security evaluation of the algorithms cryptographic (NIE; ZHANG, 2009) (DAEMEN; RIJMEN, 2013), and they are considered secure in this study.

An access control mechanism uses a cipher to provide confidentiality and must be evaluated to demonstrate that none unauthorized party can obtain the access' credentials (THILAKANATHAN *et al.*, 2014). However, this research shows this analysis is not enough to avoid the occurrence of violations.

Thus, it is essential the service monitoring and auditing to detect violations resulting from the attacks that cannot be avoided. The design and evaluation of these mechanisms are the focus of this research. In related work, it is observed the presentation of theorems to describe the security of the solutions. However, frequently, their demonstration is informally discussed (POPA *et al.*, 2011) (HWANG *et al.*, 2016).

The experimental evaluation using a prototype is suitable to analyze the performance of SCUDO, but some security aspects cannot be caught by experiments. So, the modeling and validation using CPNs are also discussed in Chapter 5. Although the formal methods have not been used in the analysis of the existing storage services, they are suitable for security evaluation based on the specified threat model. The performed evaluation demonstrates that a higher number of threats are addressed.

6.4 Limitations

The storage service proposed in this thesis focuses on the violation detection and does not address all security issues and properties, including, for example, intrusion detection and availability. Wohlin *et al.* (2012) highlight aspects that impact the validity of the experiments according to the scope of the research. For example, the *construct validity* indicates the correctness of the obtained results, showing, in this thesis, if the violations of the analyzed properties are really detected.

Security validation based on CPNs was performed to mitigate the threats to validity related to this aspect. The modeling was based on the threat model specified in Section 2.3. This threat model was extracted from literature so that the existing attacks that violate the confidentiality, integrity, freshness and write-serializability are detected by SCUDO. Nevertheless, a violation scenario, which has not yet been described in literature, may not have been caught in this research.

The *internal validity* is an aspect of validity inherent to factors that can affect the results. The experiments detailed in Section 5.1.3 exposes a limitation of SCUDO, because the detection of confidentiality violations can be bypassed if the attestations inherent to the attacks are excluded. In order to mitigate this threat, it is required the use of a trust computing solution that guarantees the logging of every transaction and that the log cannot be tampered (KO *et al.*, 2011).

In addition, the experiments performed with the prototype were limited to a few users, so that the efficiency in a complex scenario should be further evaluated, analyzing scalability issues. Although it is not possible to generalize the findings in accordance with the performed experiments, the scalability of security mechanisms, such as the broadcast encryption, is already discussed in literature (BONEH *et al.*, 2003) (POPA *et al.*, 2011). The generalization is related to *external validity* and tries to extend the results for other cases.

Last, in the ideal environment, the designed mechanisms should be native functions of the storage components of the cloud platforms. However, the applications were deployed in virtual machines and no integration with the storage component of the OpenStack, called Swift, was performed.

6.5 Future Work

This thesis proposed a cloud storage service that focuses on violations detection of security properties. This research reveals the following opportunities for future work:

- **The verification of other security properties.** Additional properties can be addressed, such as location and retrievability. Due to the legal constraints, the storage in a specific geographic location may be required, making necessary the development of solutions to verify the data location (ALBESHRI *et al.*, 2014) (JAISWAL; KUMAR, 2016). Other solutions have been proposed to identify the loss of files scarcely accessed without recovering the whole file (WANG *et al.*, 2013) (JIN *et al.*, 2018). These solutions verify the retrievability in auditing, but it is also possible to use them to improve the verification write-serializability in SCUDO.
- **The inclusion of new features.** File/folder management is a feature that can be included, allowing better organization of the files (HUO *et al.*, 2015). Thus, it is possible, for example, to rename files and change their folder. Another common feature of storage services is the version control of the stored files that allows the retrieval of previous versions of files (RAHUMED *et al.*, 2011).
- **The integration with multiple providers.** Multiple providers can be used to provide higher availability (CELESTI *et al.*, 2016). In this type of solution, the files are partitioned and distributed in different providers so that a file can be recovered even if some provider is unavailable. The broker specified in SCUDO makes the management of multiple providers feasible, controlling the partitioning and distribution of files.

- **The definition of a Security SLA.** An SLA can specify security guarantees of service, and the designed mechanisms can be used to demonstrate if SLA was violated or not. The definition of an SLA involves other aspects such as the penalties and the contingency plan to reduce the damage when a violation is detected (CARVALHO *et al.*, 2017b).
- **The use of other security mechanisms.** It is possible, for example, to analyze the use of other access control mechanisms (*e.g.*, proxy re-encryption or Attribute-Based Encryption), requiring modifications in the SCUDO.
- **The integration with the storage components of the cloud platforms.** Cloud platforms have storage components such as Swift (from OpenStack) and S3 (from Amazon), and these components can include the designed mechanisms to report the occurrence of violations.
- **The execution of new experiments.** The prototype can be updated to improve the efficiency of the designed mechanisms. For example, the log of the transactions was stored as XML files, and other approaches may reveal better results.
- **The use of a blockchain solution.** Blockchain technology has appeared as an alternative to offer transparency of data accountability in the cloud (LIANG *et al.*, 2017). In SCUDO, real-time detection requires the use of an honest broker to properly chain the attestations and verify the assurance of properties during the cloud transactions. The decentralized architecture of the blockchain can be used to register and validate the cloud transactions so that the broker is no longer needed for real-time detection. The advantage of this approach is to mitigate the collusion attacks because their success depends on compromising the entire network (GUO *et al.*, 2018).

REFERENCES

ALBESHRI, A.; BOYD, C.; Gonzalez Nieto, J. Geoproof: Proofs of geographic location for cloud computing environment. In: **Proceedings of the 32nd IEEE International Conference on Distributed Computing Systems Workshops, ICDCSW'12**. [*S.l.*: *s.n.*], 2012. p. 506–514.

ALBESHRI, A.; BOYD, C.; NIETO, J. G. A security architecture for cloud storage combining proofs of retrievability and fairness. In: **Proceedings of Cloud Computing 2012: The Third International Conference on Cloud Computing, GRIDS and Virtualization**. [*S.l.*: *s.n.*], 2012. p. 30–35.

ALBESHRI, A.; BOYD, C.; NIETO, J. G. Enhanced geoproof: improved geographic assurance for data in the cloud. **International Journal of Information Security**, v. 13, n. 2, p. 191–198, 2014.

AMAZON. **How Amazon Simple Storage Service (Amazon S3) Uses AWS KMS**. 2018. Disponível em: http://docs.aws.amazon.com/kms/latest/developerguide/services-s3.html. Acesso em: 06 ago. 2018.

AMAZON. **Managing Access Permissions to Your Amazon S3 Resources**. 2018. Disponível em: http://docs.aws.amazon.com/AmazonS3/latest/dev/s3-access-control.html. Acesso em: 06 ago. 2018.

ANDRADE, R. M.; CORREIA, M. A. S.; CARVALHO, C. A. B. d.; XIMENES, P. Evaluate location features for continuous authentication with machine learning experiments. In: **2018 International Workshop on ADVANCEs in ICT Infrastructures and Services (ADVANCE'2018)**. [*S.l.*: *s.n.*], 2018. p. 13–20.

ARDAGNA, C. A.; ASAL, R.; DAMIANI, E.; VU, Q. H. From security to assurance in the cloud: A survey. **ACM Comput. Surv.**, v. 48, n. 1, 2015.

ARDIS, M. A. Formal methods for telecommunication system requirements: A survey of standardized languages. **Annals of Software Engineering**, v. 3, n. 1, p. 157–187, 1997.

ARMANDO, A.; CARBONE, R.; COMPAGNA, L. Satmc: A sat-based model checker for security-critical systems. In: **Tools and Algorithms for the Construction and Analysis of Systems**. [*S.l.*]: Springer, 2014. p. 31–45.

BARKER, E.; BARKER, W.; BURR, W.; POLK, W.; SMID, M. Recommendation for key management part 1: General (revision 3). **NIST special publication**, v. 800, n. 57, p. 1–147, 2012.

BONEH, D.; GENTRY, C.; LYNN, B.; SHACHAM, H. Aggregate and verifiably encrypted signatures from bilinear maps. In: **International Conference on the Theory and Applications of Cryptographic Techniques**. [*S.l.*: *s.n.*], 2003. p. 416–432.

BONEH, D.; GENTRY, C.; WATERS, B. Collusion resistant broadcast encryption with short ciphertexts and private keys. In: **Advances in Cryptology – CRYPTO 2005**. [*S.l.*]: Springer Berlin Heidelberg, 2005, (Lecture Notes in Computer Science, v. 3621). p. 258–275.

BOSE, S.; PASALA, A.; Ramanujam A., D.; MURTHY, S.; MALAIYANDISAMY, G. Sla management in cloud computing: A service provider's perspective. In: **Cloud Computing: Principles and Paradigms**. [*S.l.*]: John Wiley & Sons, 2011. p. 413–436.

BROBERG, J.; BUYYA, R.; TARI, Z. Metacdn: Harnessing ‘storage clouds’ for high performance content delivery. **Journal of Network and Computer Applications**, Elsevier, v. 32, n. 5, p. 1012–1022, 2009.

BUYYA, R.; YEO, C. S.; VENUGOPAL, S.; BROBERG, J.; BRANDIC, I. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. **Future Generation computer systems**, Elsevier, v. 25, n. 6, p. 599–616, 2009.

CARVALHO, C. A. B. d.; ANDRADE, R. M. d. C.; AGOULMINE, N.; CASTRO, M. F. d. Detection of access control violations in the secure sharing of cloud storage. In: **8th International Conference on Cloud Computing and Services Science**. [*S.l.*: *s.n.*], 2018. p. 124–135.

CARVALHO, C. A. B. d.; ANDRADE, R. M. d. C.; CASTRO, M. F. d.; AGOULMINE, N. 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)**. [*S.l.*: *s.n.*], 2016. p. 17–30.

CARVALHO, C. A. B. d.; ANDRADE, R. M. d. C.; MAIA, M. E. F.; ALBUQUERQUE, D. M.; PEDROSA, E. T. O. Neutralizing vulnerabilities in android: A process and an experience report. **International Journal of Computer Science and Information Security**, v. 14, n. 3, p. 20, 2016.

CARVALHO, C. A. B. de; AGOULMINE, N.; CASTRO, M. F. de; ANDRADE, R. M. de C. How to improve monitoring and auditing security properties in cloud storage? In: **Simpósio Brasileiro de Redes de Computadores (SBRC)**. [*S.l.*: *s.n.*], 2017. v. 35, n. 559-572.

CARVALHO, C. A. B. de; ANDRADE, R. M. de C.; CASTRO, M. F. de; COUTINHO, E. F.; AGOULMINE, N. State of the art and challenges of security sla for cloud computing. **Computers & Electrical Engineering**, Elsevier, v. 59, p. 141–152, 2017.

CARVALHO, C. A. B. de; CASTRO, M. F. D.; ANDRADE, R. M. de C. Secure cloud storage service for detection of security violations. In: IEEE PRESS. **Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing**. [*S.l.*], 2017. p. 715–718.

CELESTI, A.; FAZIO, M.; VILLARI, M.; PULIAFITO, A. Adding long-term availability, obfuscation, and encryption to multi-cloud storage systems. **Journal of Network and Computer Applications**, v. 59, p. 208–218, 2016.

CHANG, S. E.; JANG, Y.-T. J.; SHEN, W.-C.; SU, W.-C. Cocktail: a service-oriented cloud storage architecture for enhancing service quality. **International Journal of High Performance Computing and Networking**, v. 9, n. 1-2, p. 19–30, 2016.

CHENG, A.; CHRISTENSEN, S.; MORTENSEN, K. H. Model checking coloured petri nets-exploiting strongly connected components. **DAIMI report series**, n. 519, 1997.

CLARKE, E. M.; EMERSON, E. A.; SISTLA, A. P. Automatic verification of finite-state concurrent systems using temporal logic specifications. **ACM Transactions on Programming Languages and Systems (TOPLAS)**, v. 8, n. 2, p. 244–263, 1986.

CLARKE, E. M.; WING, J. M. Formal methods: State of the art and future directions. **ACM Computing Surveys (CSUR)**, v. 28, n. 4, p. 626–643, 1996.

CSA. The notorious nine: Cloud computing top threats in 2013. **Top Threats Working Group**, February 2013.

DAEMEN, J.; RIJMEN, V. **The design of Rijndael: AES-the advanced encryption standard**. [*S.l.*]: Springer Science & Business Media, 2013.

DOLEV, D.; YAO, A. C. On the security of public key protocols. **Information Theory, IEEE Transactions on**, v. 29, n. 2, p. 198–208, 1983.

FENG, J.; CHEN, Y.; SUMMERVILLE, D. H. A fair multi-party non-repudiation scheme for storage clouds. In: **Collaboration Technologies and Systems (CTS), 2011 International Conference on**. [*S.l.*: *s.n.*], 2011. p. 457–465.

GUO, R.; SHI, H.; ZHAO, Q.; ZHENG, D. Secure attribute-based signature scheme with multiple authorities for blockchain in electronic health records systems. **IEEE Access**, IEEE, v. 6, p. 11676–11686, 2018.

HABIB, S. M.; RIES, S.; MÜHLHÄUSER, M. Towards a trust management system for cloud computing. In: **Trust, Security and Privacy in Computing and Communications (TrustCom), 2011 IEEE 10th International Conference on**. [*S.l.*: *s.n.*], 2011. p. 933–939.

HUO, J.; QU, H.; WU, L. Design and implementation of private cloud storage platform based on openstack. In: IEEE. **2015 IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity)**. [*S.l.*], 2015. p. 1098–1101.

HWANG, G.-H.; HUANG, W.-S.; PENG, J.-Z. Real-time proof of violation for cloud storage. In: **Cloud Computing Technology and Science (CloudCom), 2014 IEEE 6th International Conference on**. [*S.l.*: *s.n.*], 2014. p. 394–399.

HWANG, G.-H.; HUANG, W.-S.; PENG, J.-Z.; LIN, Y.-W. Fulfilling mutual nonrepudiation for cloud storage. **Concurrency and Computation: Practice and Experience**, v. 28, n. 3, p. 583–599, 2016.

JAISWAL, C.; KUMAR, V. Igod: identification of geolocation of cloud datacenters. **Journal of Information Security and Applications**, Elsevier, v. 27, p. 85–102, 2016.

JENSEN, K.; KRISTENSEN, L. M. **Coloured Petri nets: modelling and validation of concurrent systems**. [*S.l.*]: Springer Science & Business Media, 2009.

JENSEN, K.; KRISTENSEN, L. M.; WELLS, L. Coloured petri nets and cpn tools for modelling and validation of concurrent systems. **International Journal on Software Tools for Technology Transfer**, v. 9, n. 3-4, p. 213–254, 2007.

JIANG, W.; WANG, Z.; LIU, L.; GAO, N. Towards efficient update of access control policy for cryptographic cloud storage. In: **International Conference on Security and Privacy in Communication Systems**. [*S.l.*: *s.n.*], 2014. p. 341–356.

JIN, H.; ZHOU, K.; JIANG, H.; LEI, D.; WEI, R.; LI, C. Full integrity and freshness for cloud data. **Future Generation Computer Systems**, v. 80, p. 640–652, 2018.

KALLAHALLA, M.; RIEDEL, E.; SWAMINATHAN, R.; WANG, Q.; FU, K. Plutus: Scalable secure file sharing on untrusted storage. In: **Fast**. [*S.l.*: *s.n.*], 2003. v. 3, p. 29–42.

KO, R. K.; JAGADPRAMANA, P.; MOWBRAY, M.; PEARSON, S.; KIRCHBERG, M.; LIANG, Q.; LEE, B. S. Trustcloud: A framework for accountability and trust in cloud computing. In: **Services (SERVICES), 2011 IEEE World Congress on**. [*S.l.*: *s.n.*], 2011. p. 584–588.

LI, M.; YU, S.; ZHENG, Y.; REN, K.; LOU, W. Scalable and secure sharing of personal health records in cloud computing using attribute-based encryption. **IEEE transactions on parallel and distributed systems**, v. 24, n. 1, p. 131–143, 2013.

LIANG, X.; SHETTY, S.; TOSH, D.; KAMHOUA, C.; KWIAT, K.; NJILLA, L. Provchain: A blockchain-based data provenance architecture in cloud environment with enhanced privacy and availability. In: IEEE PRESS. **Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing**. [*S.l.*], 2017. p. 468–477.

LIU, Q.; WANG, G.; WU, J. Secure and privacy preserving keyword searching for cloud storage services. **Journal of network and computer applications**, v. 35, n. 3, p. 927–933, 2012.

LOO, A. Distributed multiple selection algorithm for peer-to-peer systems. **Journal of Systems and Software**, Elsevier, v. 78, n. 3, p. 234–248, 2005.

LOPEZ, J.; ROMAN, R.; ALCARAZ, C. Analysis of security threats, requirements, technologies and standards in wireless sensor networks. In: **Foundations of Security Analysis and Design V**. [*S.l.*]: Springer, 2009. p. 289–338.

LOWE, G. An attack on the needham-schroeder public-key authentication protocol. **Information processing letters**, v. 56, n. 3, p. 131–133, 1995.

LUNA, J.; SURI, N.; IORGA, M.; KARMEL, A. Leveraging the potential of cloud security service-level agreements through standards. **IEEE Cloud Computing Magazine**, v. 2, n. 3, p. 32 – 40, 2015.

MA, H.; ZHANG, R. Secure cloud storage for dynamic group: How to achieve identity privacy-preserving and privilege control. In: **International Conference on Network and System Security**. [*S.l.*: *s.n.*], 2015. p. 254–267.

MODI, C.; PATEL, D.; BORISANIYA, B.; PATEL, A.; RAJARAJAN, M. A survey on security issues and solutions at different layers of cloud computing. **The journal of supercomputing**, v. 63, n. 2, p. 561–592, 2013.

NEEDHAM, R. M.; SCHROEDER, M. D. Using encryption for authentication in large networks of computers. **Communications of the ACM**, ACM, v. 21, n. 12, p. 993–999, 1978.

NIE, T.; ZHANG, T. A study of des and blowfish encryption algorithm. In: **TENCON 2009-2009 IEEE Region 10 Conference**. [*S.l.*: *s.n.*], 2009. p. 1–4.

POMMEREAU, F. Algebras of coloured petri nets. **LAP LAMBERT Academic Publishing**, 2010.

POPA, R. A.; LORCH, J. R.; MOLNAR, D.; WANG, H. J.; ZHUANG, L. Enabling security in cloud storage slas with cloudproof. In: **Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference, USENIXATC'11**. [*S.l.*: *s.n.*], 2011. p. 355–378.

RAHUMED, A.; CHEN, H. C.; TANG, Y.; LEE, P. P.; LUI, J. C. A secure cloud backup system with assured deletion and version control. In: IEEE. **2011 40th International Conference on Parallel Processing Workshops**. [*S.l.*], 2011. p. 160–167.

RONG, C.; NGUYEN, S. T.; JAATUN, M. G. Beyond lightning: a survey on security challenges in cloud computing. **Computers and Electrical Engineering**, v. 39, n. 1, p. 47–54, 2013.

RUMBAUGH, J.; JACOBSON, I.; BOOCH, G. **Unified modeling language reference manual, the**. [*S.l.*]: Pearson Higher Education, 2004.

SARIPALLI, P.; WALTERS, B. Quirc: A quantitative impact and risk assessment framework for cloud security. In: **Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on**. [*S.l.*: *s.n.*], 2010. p. 280–288.

SEIFI, Y. **Formal Analysis of Security Properties in Trusted Computing Protocols**. Tese (Doutorado) — Queensland University of Technology, 2014.

STALLINGS, W. **Cryptography and network security: principles and practices**. 7. ed. [*S.l.*]: Pearson, 2016.

STAMOU, K.; AUBERT, J.; GATEAU, B.; MORIN, J.-H. Preliminary requirements on trusted third parties for service transactions in cloud environments. In: **46th Hawaii International Conference on System Sciences**. [*S.l.*: *s.n.*], 2013. p. 4976–4983.

SUN, Y.; ZHANG, J.; XIONG, Y.; ZHU, G. Data security and privacy in cloud computing. **International Journal of Distributed Sensor Networks**, v. 2014, p. 1–9, 2014.

TASSANAVIBOON, A.; GONG, G. Oauth and abe based authorization in semi-trusted cloud computing: aauth. In: **Proceedings of the second international workshop on Data intensive computing in the clouds**. [*S.l.*: *s.n.*], 2011. p. 41–50.

THILAKANATHAN, D.; CHEN, S.; NEPAL, S.; CALVO, R. A. Secure data sharing in the cloud. In: **Security, Privacy and Trust in Cloud Systems**. [*S.l.*]: Springer, 2014. p. 45–72.

TIWARI, D.; GANGADHARAN, G. A novel secure cloud storage architecture combining proof of retrievability and revocation. In: **Advances in Computing, Communications and Informatics (ICACCI), 2015 International Conference on**. [*S.l.*: *s.n.*], 2015. p. 438–445.

TIWARI, D.; GANGADHARAN, G. Secure sharing of data in cloud computing. In: **International Symposium on Security in Computing and Communication**. [*S.l.*: *s.n.*], 2015. p. 24–35.

TRAPERO, R.; MODIC, J.; STOPAR, M.; TAHA, A.; SURI, N. A novel approach to manage cloud security sla incidents. **Future Generation Computer Systems**, v. 72, p. 193 – 205, 2017.

VAUDENAY, S.; VUAGNOUX, M. Passive–only key recovery attacks on rc4. In: **International Workshop on Selected Areas in Cryptography**. [*S.l.*: *s.n.*], 2007. p. 344–359.

VILLANI, E.; MIYAGI, P. E. A hybrid petri net modeling approach for hvac systems in intelligent buildings. **Sba: Controle & Automação Sociedade Brasileira de Automatica**, v. 15, n. 2, p. 135–148, 2004.

WANG, C.; CHOW, S. S.; WANG, Q.; REN, K.; LOU, W. Privacy-preserving public auditing for secure cloud storage. **IEEE Transactions on computers**, v. 62, n. 2, p. 362–375, 2013.

WEI, B.; FEDAK, G.; CAPPELLO, F. Towards efficient data distribution on computational desktop grids with bittorrent. **Future Generation Computer Systems**, Elsevier, v. 23, n. 8, p. 983–989, 2007.

WOHLIN, C.; RUNESON, P.; HÖST, M.; OHLSSON, M. C.; REGNELL, B.; WESSLÉN, A. **Experimentation in software engineering**. [*S.l.*]: Springer Science & Business Media, 2012.

WORKU, S. G.; XU, C.; ZHAO, J.; HE, X. Secure and efficient privacy-preserving public auditing scheme for cloud storage. **Computers & Electrical Engineering**, v. 40, n. 5, p. 1703–1713, 2014.

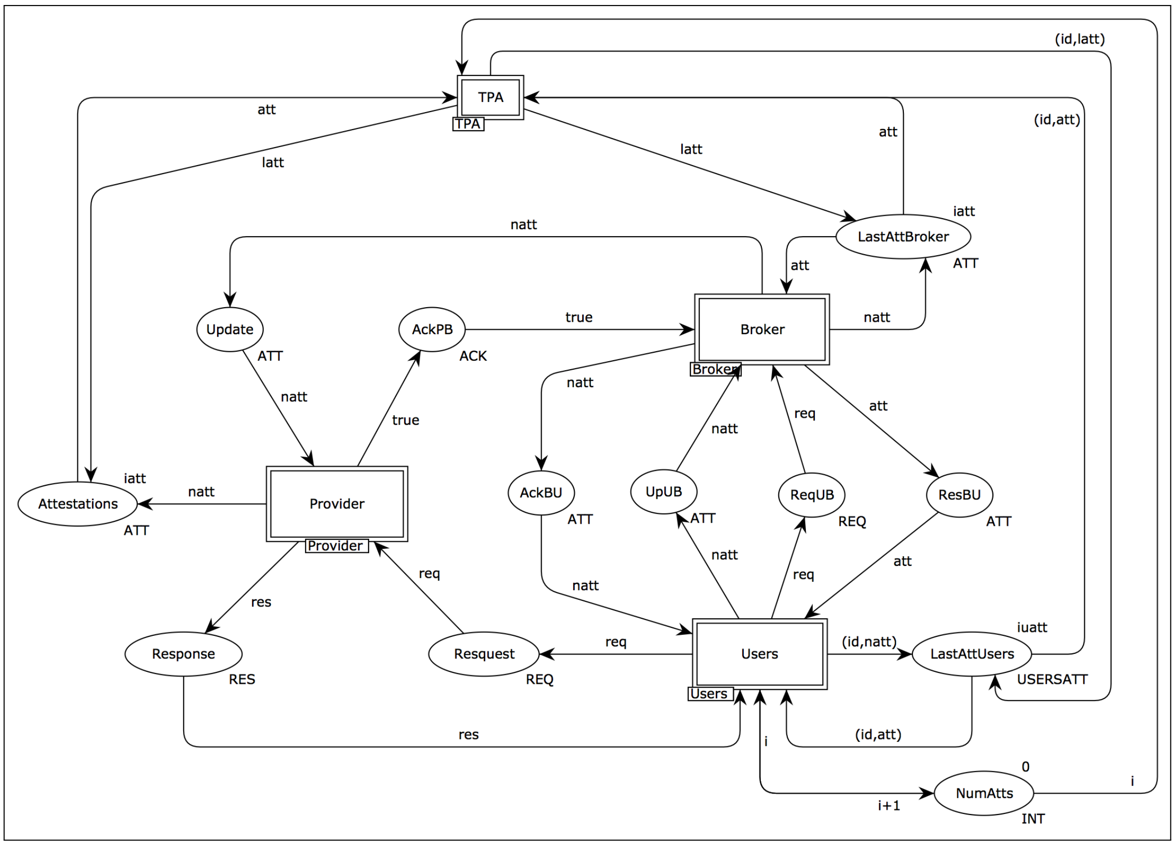
XIA, Z.; WANG, X.; ZHANG, L.; QIN, Z.; SUN, X.; REN, K. A privacy-preserving and copy-deterrence content-based image retrieval scheme in cloud computing. **IEEE Transactions on Information Forensics and Security**, v. 11, n. 11, p. 2594–2608, 2016.

YANG, K.; JIA, X. Data storage auditing service in cloud computing: challenges, methods and opportunities. **World Wide Web**, v. 15, n. 4, p. 409–428, 2012.

APPENDIX A – CPN MODEL

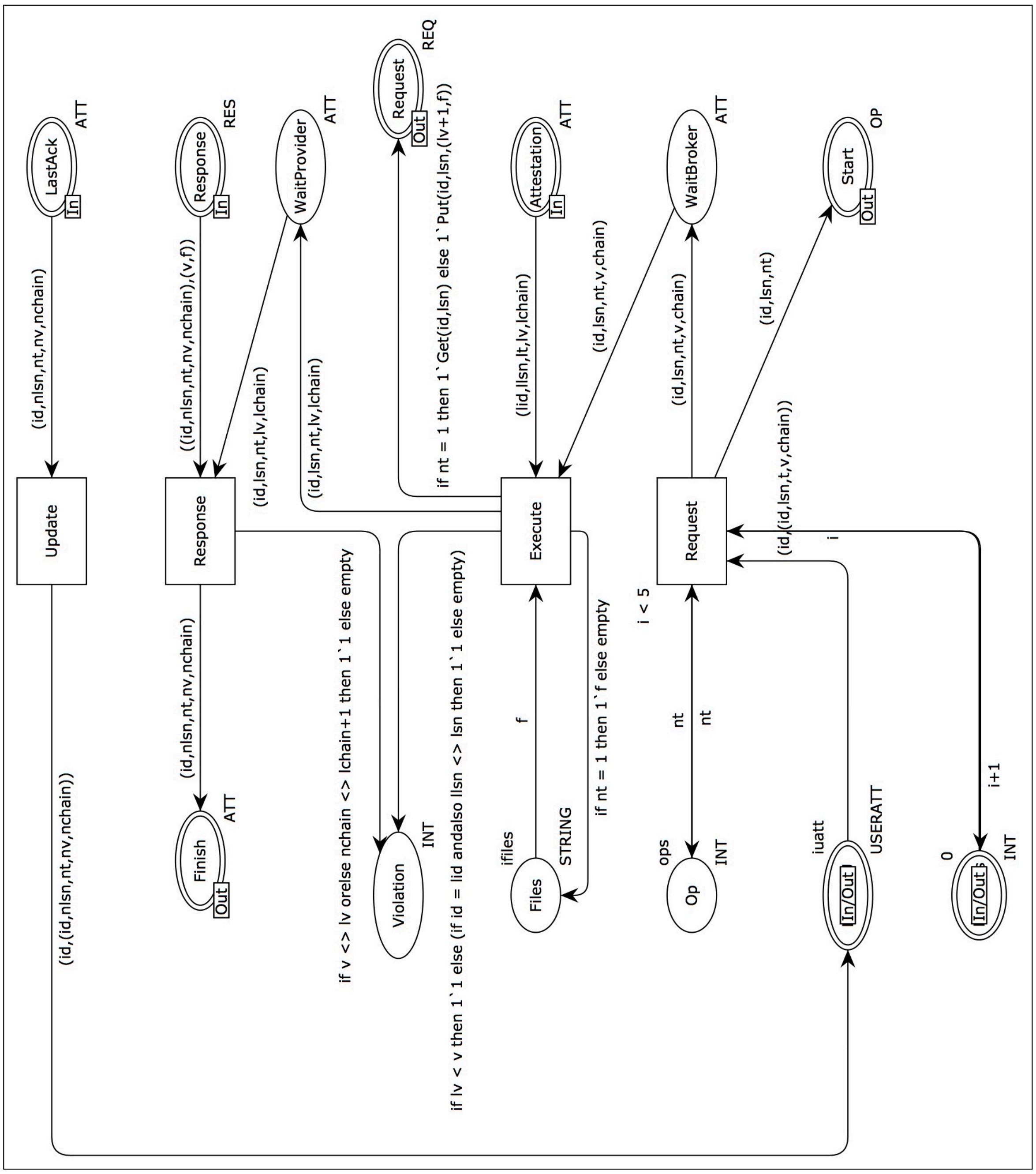
This appendix presents figures exported from the CPN modeling of SCUDO. Figure 26 displays the messages exchanged during the execution of each transaction and the sending of the attestations for auditing. In addition, Figures 27, 28 and 29 detail the actions performed, respectively, by the users, the broker and the provider, according to the protocols for execution of the transactions. Last, Figure 30 describes how the auditor verifies the chain of attestations.

Figure 26 – Overview of the modeled CPN



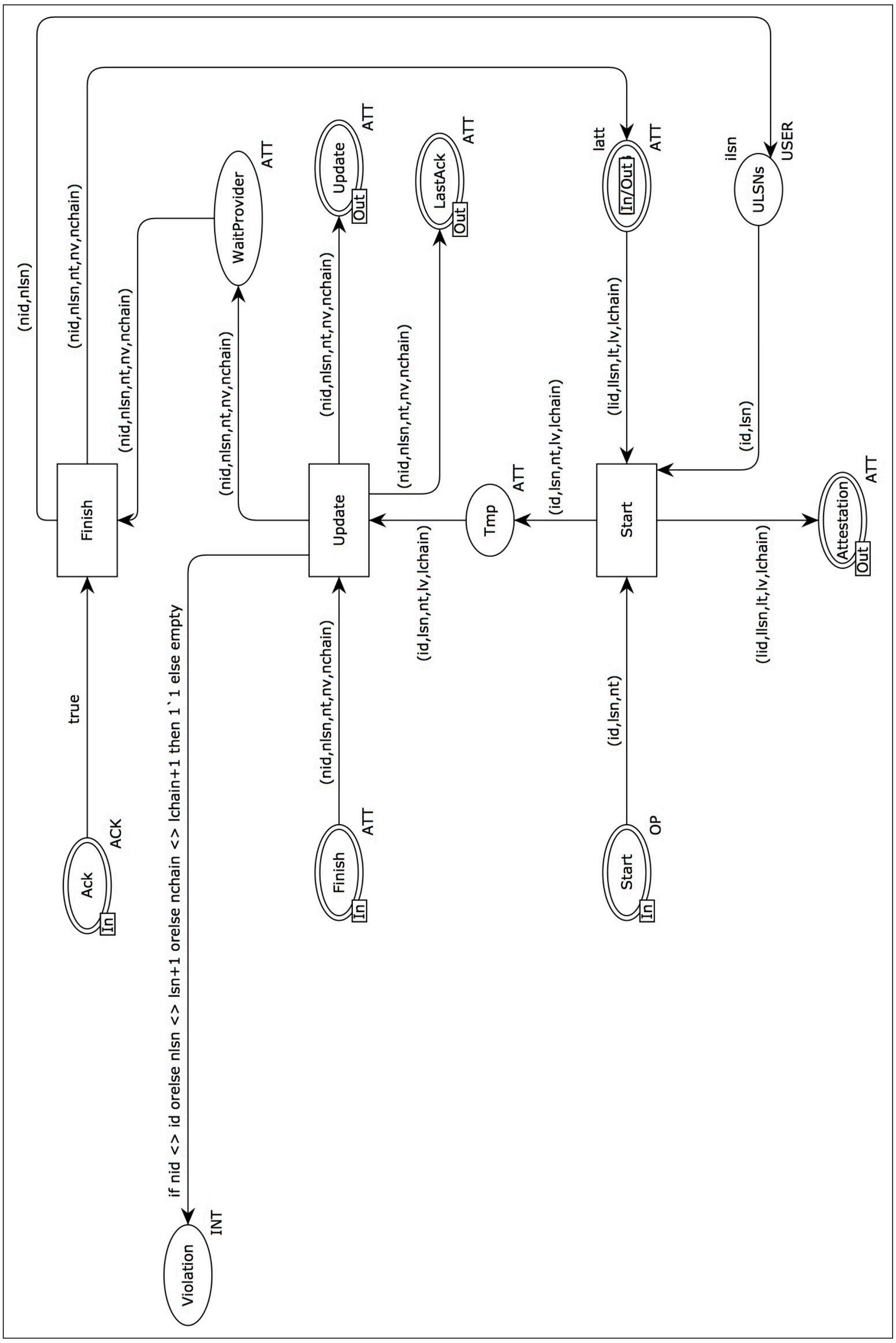
Source – The author.

Figure 27 – The sub-module of the users



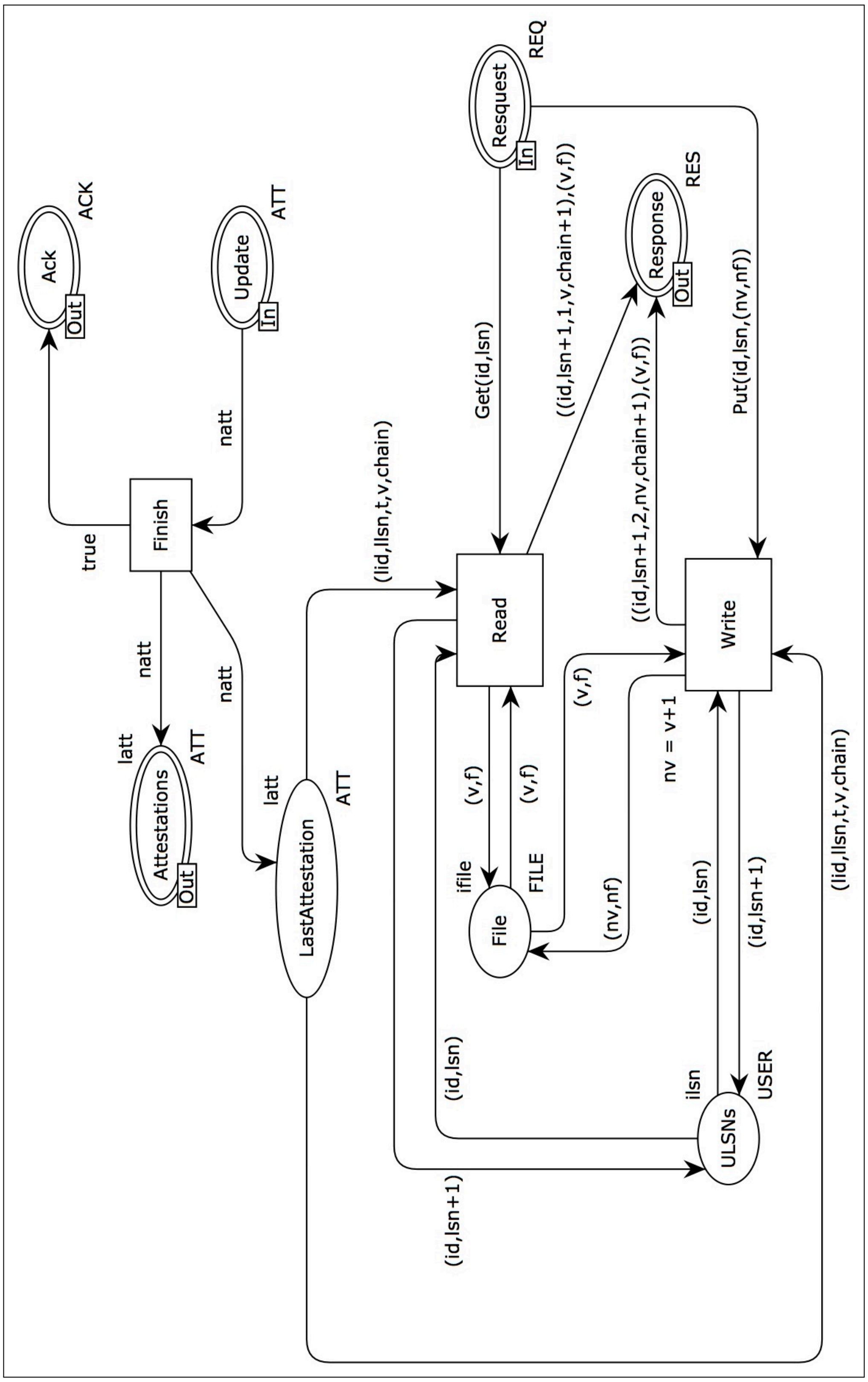
Source – The author.

Figure 28 – The sub-module of the broker



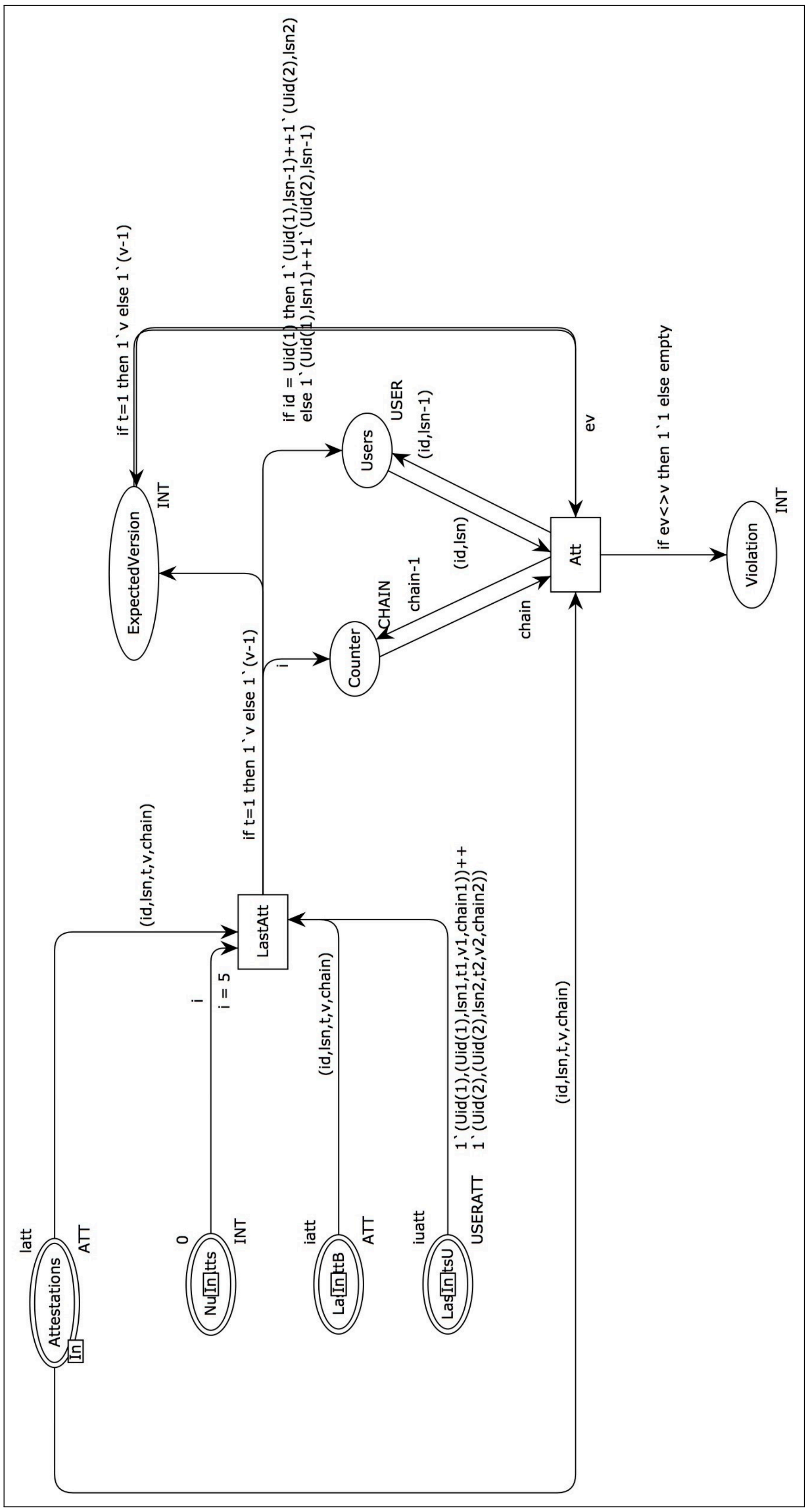
Source – The author.

Figure 29 – The sub-module of the provider



Source – The author.

Figure 30 – The sub-module of TPA


Attestations
In
Iatt
ATT
Nu In tts
0
INT
La In ttB
iatt
ATT
Las In tsU
iuatt
USERATT
LastAtt
(id,lsn,t,v,chain)
i
i = 5
(id,lsn,t,v,chain)
1`(Uid(1),(Uid(1),lsn1,t1,v1,chain1))++
1`(Uid(2),(Uid(2),lsn2,t2,v2,chain2))
(id,lsn,t,v,chain)
if t=1 then 1`v else 1`(v-1)
i
Counter
CHAIN
chain-1
chain
ExpectedVersion
INT
if t=1 then 1`v else 1`(v-1)
if id = Uid(1) then 1`(Uid(1),lsn-1)++1`(Uid(2),lsn2)
else 1`(Uid(1),lsn1)++1`(Uid(2),lsn-1)
Users
USER
(id,lsn-1)
(id,lsn)
Att
ev
if ev<>v then 1`1 else empty
Violation
INT

Source – The author.

APPENDIX B – EXPERIMENTS RESULTS

This appendix presents data of the experiments performed to analyze the performance of the developed prototype.

B.1 Summary of the results

Table 7 displays an overview of the performed experiments, extracting the average time for the execution of each type of transaction. In addition, the time related to cost of the overhead, the cryptographic functions and the network are collected. The network cost involves the time to download or to upload the file and is mainly influenced by file size. When analyzing the transactions for permissions update, the time to create and distribute new keys and the time related to the revocation of the permissions are obtained. The lazy revocation was not developed due to the loss of security, discussed in Section 5.1.3. Thus, the revocation involves a higher cost, including the time for downloading, re-encrypting and uploading the files.

Table 7 – Average time of each transaction

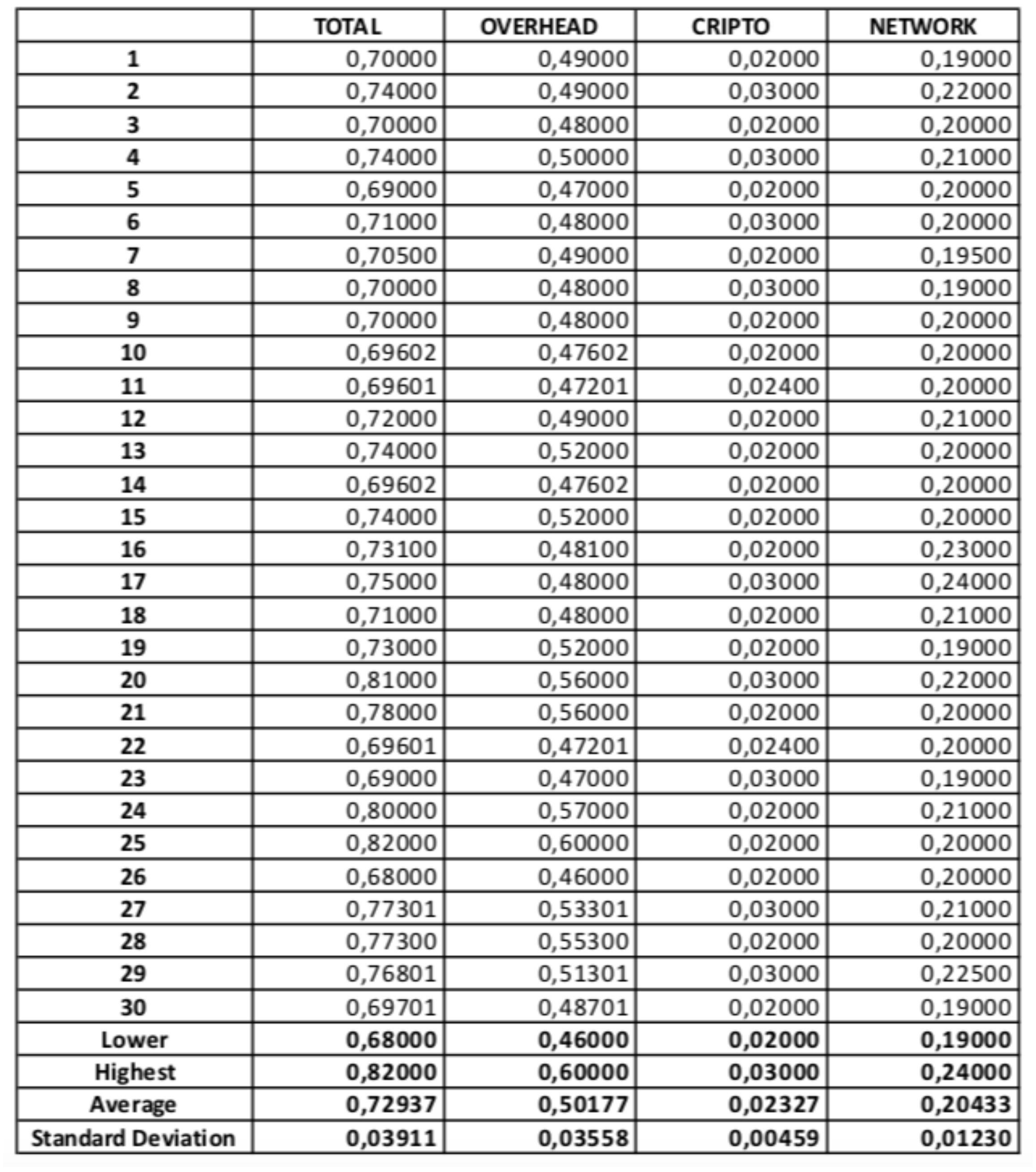
READING	8KB	32KB	128KB	512KB	1024KB
Overhead	0,50177	0,49770	0,48547	0,46937	0,47317
Cryptograph	0,02327	0,03290	0,04053	0,08450	0,14493
Network	0,20433	0,27007	0,46384	0,52554	0,66257
Total	0,72937	0,80067	0,98984	1,07941	1,28067
WRITING	**8KB**	**32KB**	**128KB**	**512KB**	**1024KB**
Overhead	0,70371	0,69187	0,70851	0,69857	0,68000
Network	0,11370	0,23187	0,69524	2,38792	4,45379
Cryptograph	0,18480	0,19177	0,19460	0,19957	0,20534
Total	1,00221	1,11551	1,59835	3,28606	5,33913
UPDATING PERMISSIONS	**8KB**	**32KB**	**128KB**	**512KB**	**1024KB**
New Keys	1,32941	1,39785	1,36261	1,35738	1,41981
Revocation	0,32087	0,43061	1,11154	2,73653	4,90954
Overhead	0,79744	0,79241	0,81694	0,79224	0,80204
Total	2,44772	2,63420	3,29110	4,88615	7,13140

Source – The author.

B.2 Reading Transactions

Tables 8, 9, 10, 11 and 12 detail the time of the execution of reading transactions, considering each one of the thirty repetitions and according to the file size.

Table 8 – Reading 8KB files



	TOTAL	OVERHEAD	CRIPTO	NETWORK
1	0,70000	0,49000	0,02000	0,19000
2	0,74000	0,49000	0,03000	0,22000
3	0,70000	0,48000	0,02000	0,20000
4	0,74000	0,50000	0,03000	0,21000
5	0,69000	0,47000	0,02000	0,20000
6	0,71000	0,48000	0,03000	0,20000
7	0,70500	0,49000	0,02000	0,19500
8	0,70000	0,48000	0,03000	0,19000
9	0,70000	0,48000	0,02000	0,20000
10	0,69602	0,47602	0,02000	0,20000
11	0,69601	0,47201	0,02400	0,20000
12	0,72000	0,49000	0,02000	0,21000
13	0,74000	0,52000	0,02000	0,20000
14	0,69602	0,47602	0,02000	0,20000
15	0,74000	0,52000	0,02000	0,20000
16	0,73100	0,48100	0,02000	0,23000
17	0,75000	0,48000	0,03000	0,24000
18	0,71000	0,48000	0,02000	0,21000
19	0,73000	0,52000	0,02000	0,19000
20	0,81000	0,56000	0,03000	0,22000
21	0,78000	0,56000	0,02000	0,20000
22	0,69601	0,47201	0,02400	0,20000
23	0,69000	0,47000	0,03000	0,19000
24	0,80000	0,57000	0,02000	0,21000
25	0,82000	0,60000	0,02000	0,20000
26	0,68000	0,46000	0,02000	0,20000
27	0,77301	0,53301	0,03000	0,21000
28	0,77300	0,55300	0,02000	0,20000
29	0,76801	0,51301	0,03000	0,22500
30	0,69701	0,48701	0,02000	0,19000
Lower	**0,68000**	**0,46000**	**0,02000**	**0,19000**
Highest	**0,82000**	**0,60000**	**0,03000**	**0,24000**
Average	**0,72937**	**0,50177**	**0,02327**	**0,20433**
Standard Deviation	**0,03911**	**0,03558**	**0,00459**	**0,01230**

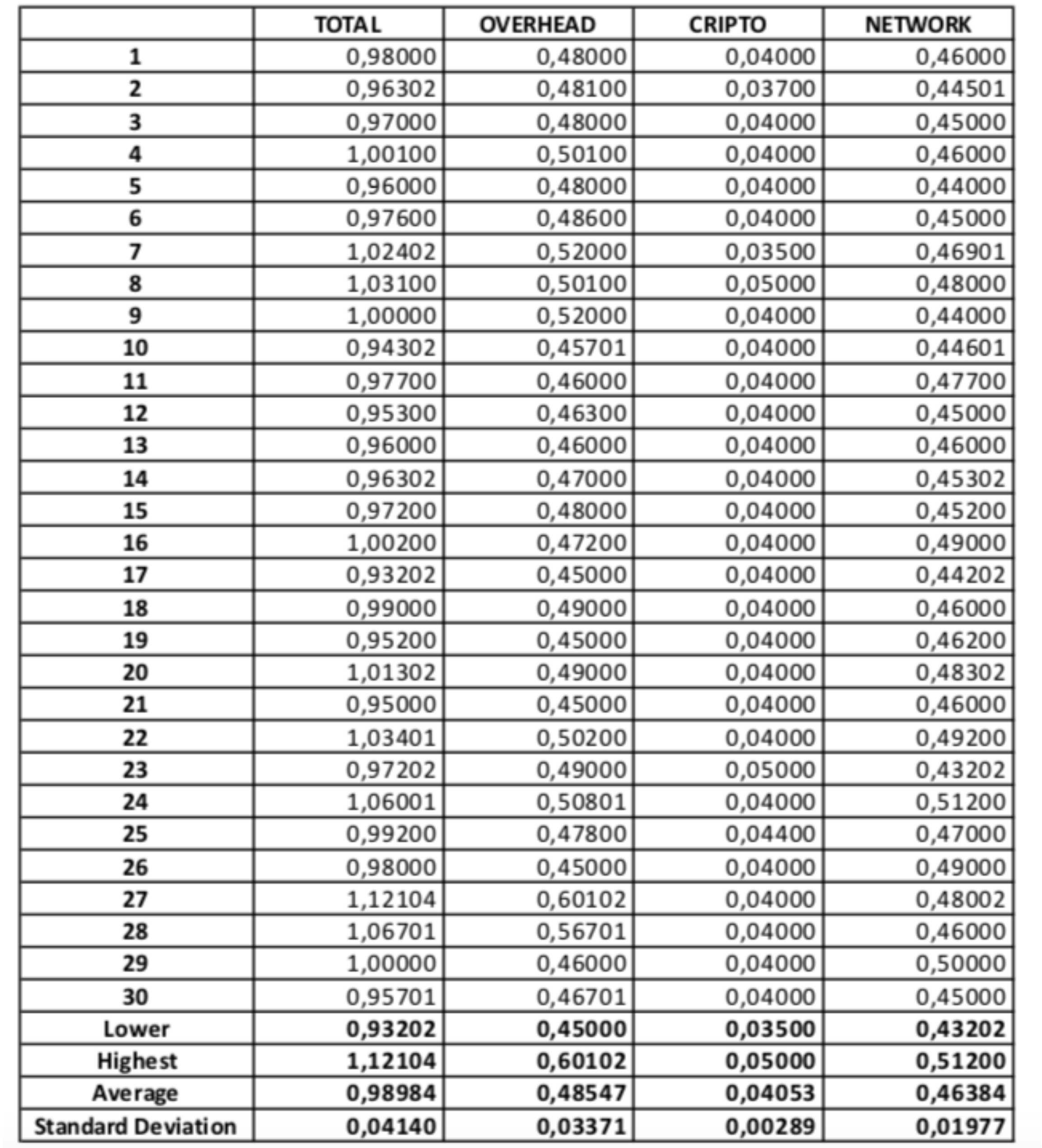
Source – The author.

Table 9 – Reading 32KB files

	TOTAL	OVERHEAD	CRIPTO	NETWORK
1	0,77000	0,48000	0,03000	0,26000
2	0,83200	0,52200	0,03000	0,28000
3	0,82000	0,52000	0,03000	0,27000
4	0,80200	0,48100	0,03100	0,29000
5	0,78000	0,48000	0,04000	0,26000
6	0,78700	0,49700	0,03000	0,26000
7	0,80100	0,49100	0,04000	0,27000
8	0,86000	0,55000	0,04000	0,27000
9	0,89000	0,58000	0,03000	0,28000
10	0,87000	0,54000	0,04000	0,29000
11	0,78000	0,47000	0,04000	0,27000
12	0,76000	0,47000	0,03000	0,26000
13	0,79000	0,49000	0,03000	0,27000
14	0,81301	0,50300	0,03000	0,28000
15	0,80500	0,51900	0,03000	0,25600
16	0,76702	0,47702	0,03000	0,26000
17	0,80000	0,51000	0,03000	0,26000
18	0,76000	0,46000	0,03000	0,27000
19	0,77001	0,48401	0,03600	0,25000
20	0,73201	0,45201	0,03000	0,25000
21	0,78000	0,49000	0,03000	0,26000
22	0,80000	0,46000	0,03000	0,31000
23	0,78000	0,49000	0,03000	0,26000
24	0,78000	0,48000	0,03000	0,27000
25	0,87000	0,55000	0,04000	0,28000
26	0,78802	0,48501	0,03000	0,27301
27	0,78000	0,47000	0,04000	0,27000
28	0,81000	0,51000	0,03000	0,27000
29	0,82000	0,51000	0,03000	0,28000
30	0,82300	0,51000	0,04000	0,27300
Lower	**0,73201**	**0,45201**	**0,03000**	**0,25000**
Highest	**0,89000**	**0,58000**	**0,04000**	**0,31000**
Average	**0,80067**	**0,49770**	**0,03290**	**0,27007**
Standard Deviation	**0,03604**	**0,02975**	**0,00449**	**0,01275**

Source – The author.

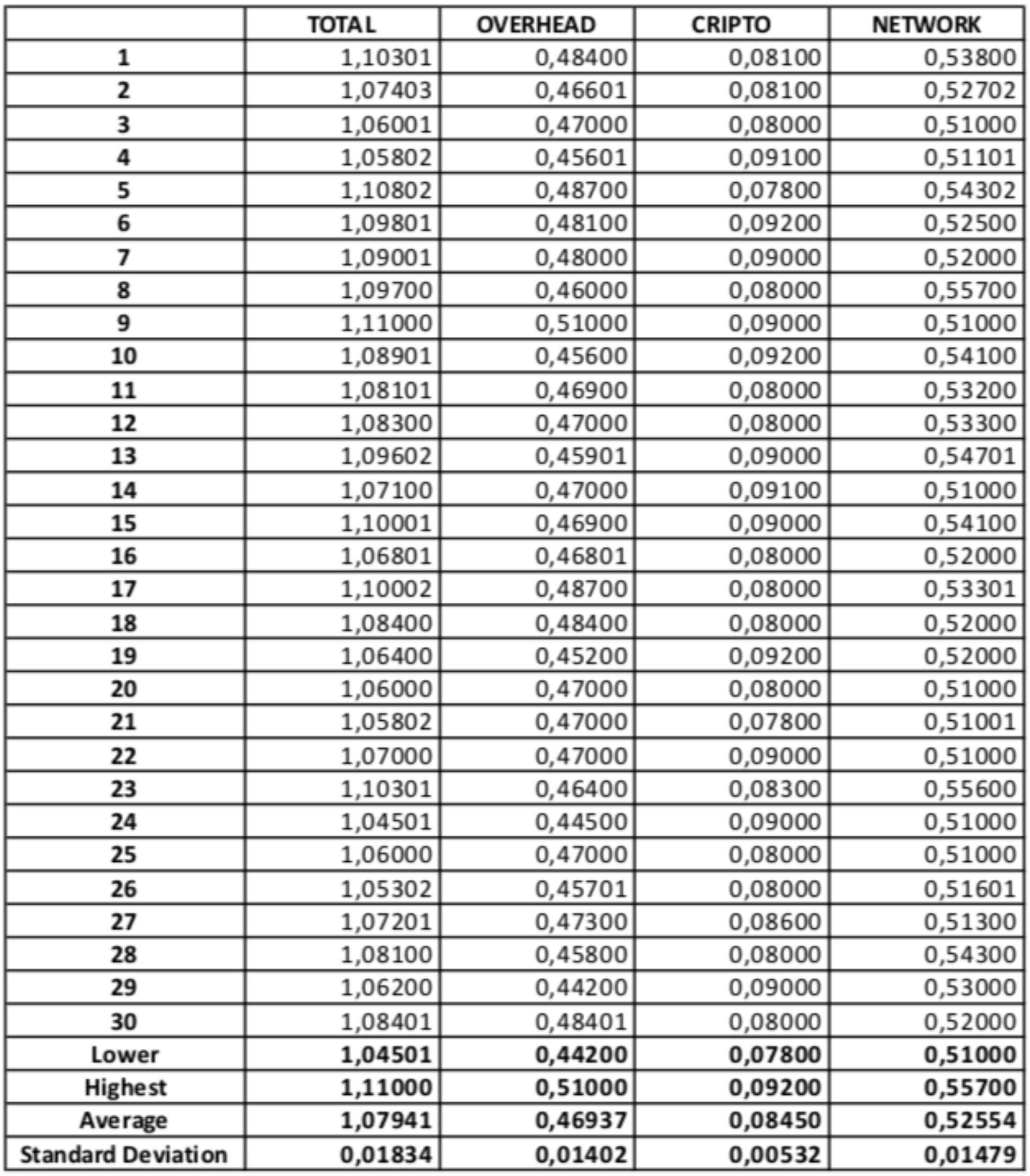
Table 10 – Reading 128KB files



	TOTAL	OVERHEAD	CRIPTO	NETWORK
1	0,98000	0,48000	0,04000	0,46000
2	0,96302	0,48100	0,03700	0,44501
3	0,97000	0,48000	0,04000	0,45000
4	1,00100	0,50100	0,04000	0,46000
5	0,96000	0,48000	0,04000	0,44000
6	0,97600	0,48600	0,04000	0,45000
7	1,02402	0,52000	0,03500	0,46901
8	1,03100	0,50100	0,05000	0,48000
9	1,00000	0,52000	0,04000	0,44000
10	0,94302	0,45701	0,04000	0,44601
11	0,97700	0,46000	0,04000	0,47700
12	0,95300	0,46300	0,04000	0,45000
13	0,96000	0,46000	0,04000	0,46000
14	0,96302	0,47000	0,04000	0,45302
15	0,97200	0,48000	0,04000	0,45200
16	1,00200	0,47200	0,04000	0,49000
17	0,93202	0,45000	0,04000	0,44202
18	0,99000	0,49000	0,04000	0,46000
19	0,95200	0,45000	0,04000	0,46200
20	1,01302	0,49000	0,04000	0,48302
21	0,95000	0,45000	0,04000	0,46000
22	1,03401	0,50200	0,04000	0,49200
23	0,97202	0,49000	0,05000	0,43202
24	1,06001	0,50801	0,04000	0,51200
25	0,99200	0,47800	0,04400	0,47000
26	0,98000	0,45000	0,04000	0,49000
27	1,12104	0,60102	0,04000	0,48002
28	1,06701	0,56701	0,04000	0,46000
29	1,00000	0,46000	0,04000	0,50000
30	0,95701	0,46701	0,04000	0,45000
Lower	**0,93202**	**0,45000**	**0,03500**	**0,43202**
Highest	**1,12104**	**0,60102**	**0,05000**	**0,51200**
Average	**0,98984**	**0,48547**	**0,04053**	**0,46384**
Standard Deviation	**0,04140**	**0,03371**	**0,00289**	**0,01977**

Source – The author.

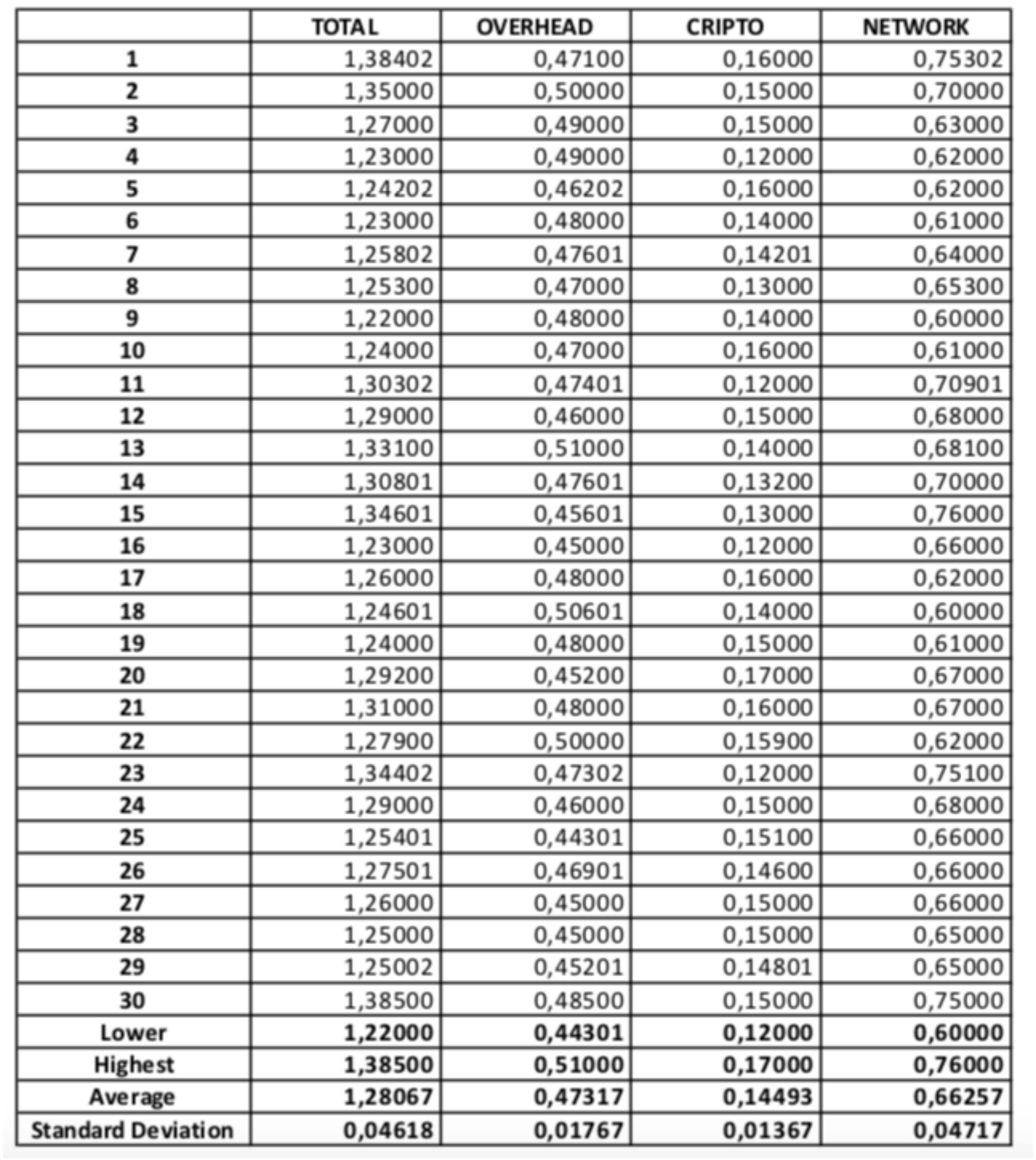
Table 11 – Reading 512KB files



	TOTAL	OVERHEAD	CRIPTO	NETWORK
1	1,10301	0,48400	0,08100	0,53800
2	1,07403	0,46601	0,08100	0,52702
3	1,06001	0,47000	0,08000	0,51000
4	1,05802	0,45601	0,09100	0,51101
5	1,10802	0,48700	0,07800	0,54302
6	1,09801	0,48100	0,09200	0,52500
7	1,09001	0,48000	0,09000	0,52000
8	1,09700	0,46000	0,08000	0,55700
9	1,11000	0,51000	0,09000	0,51000
10	1,08901	0,45600	0,09200	0,54100
11	1,08101	0,46900	0,08000	0,53200
12	1,08300	0,47000	0,08000	0,53300
13	1,09602	0,45901	0,09000	0,54701
14	1,07100	0,47000	0,09100	0,51000
15	1,10001	0,46900	0,09000	0,54100
16	1,06801	0,46801	0,08000	0,52000
17	1,10002	0,48700	0,08000	0,53301
18	1,08400	0,48400	0,08000	0,52000
19	1,06400	0,45200	0,09200	0,52000
20	1,06000	0,47000	0,08000	0,51000
21	1,05802	0,47000	0,07800	0,51001
22	1,07000	0,47000	0,09000	0,51000
23	1,10301	0,46400	0,08300	0,55600
24	1,04501	0,44500	0,09000	0,51000
25	1,06000	0,47000	0,08000	0,51000
26	1,05302	0,45701	0,08000	0,51601
27	1,07201	0,47300	0,08600	0,51300
28	1,08100	0,45800	0,08000	0,54300
29	1,06200	0,44200	0,09000	0,53000
30	1,08401	0,48401	0,08000	0,52000
Lower	**1,04501**	**0,44200**	**0,07800**	**0,51000**
Highest	**1,11000**	**0,51000**	**0,09200**	**0,55700**
Average	**1,07941**	**0,46937**	**0,08450**	**0,52554**
Standard Deviation	**0,01834**	**0,01402**	**0,00532**	**0,01479**

Source – The author.

Table 12 – Reading 1024KB files



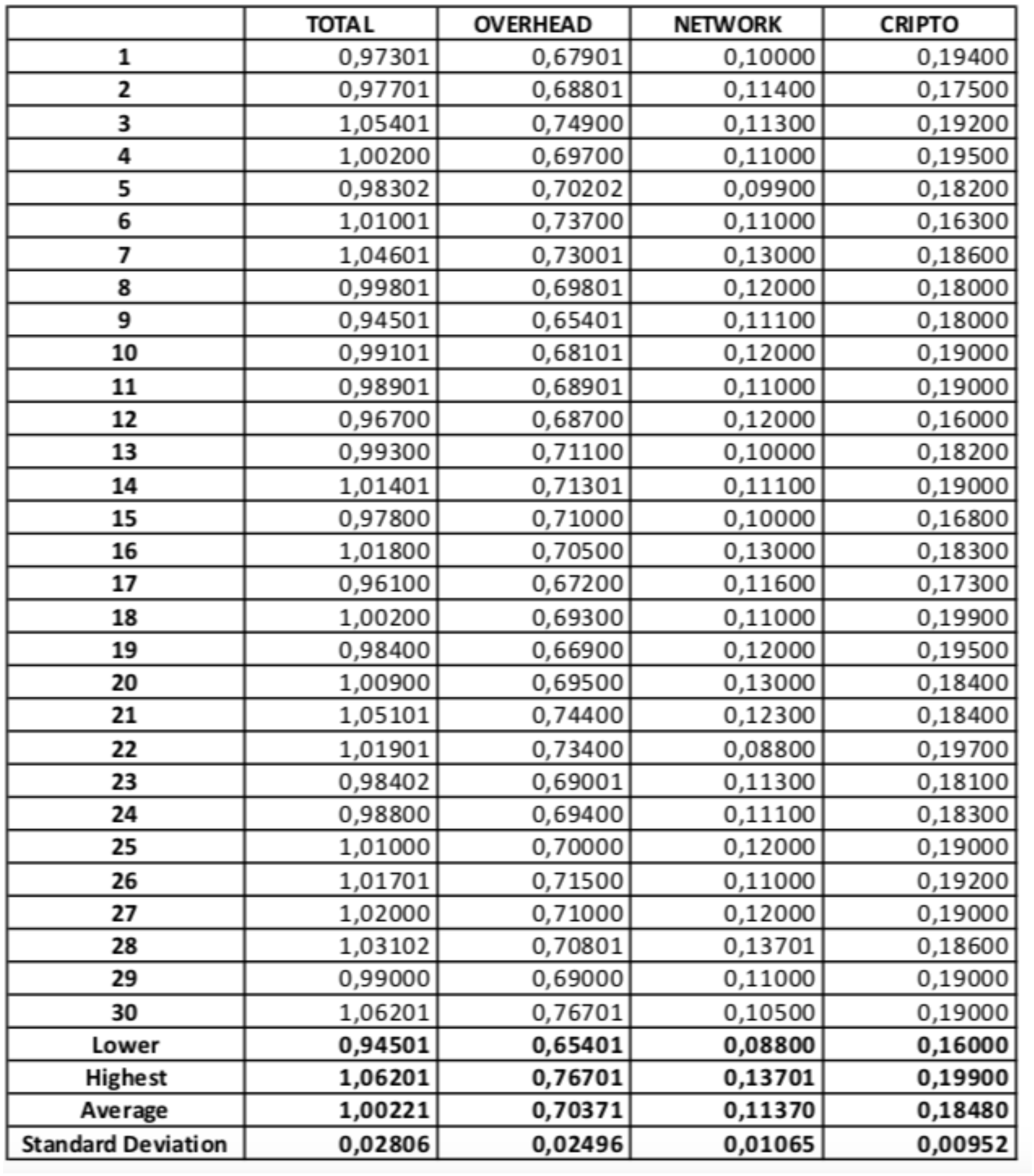
	TOTAL	OVERHEAD	CRIPTO	NETWORK
1	1,38402	0,47100	0,16000	0,75302
2	1,35000	0,50000	0,15000	0,70000
3	1,27000	0,49000	0,15000	0,63000
4	1,23000	0,49000	0,12000	0,62000
5	1,24202	0,46202	0,16000	0,62000
6	1,23000	0,48000	0,14000	0,61000
7	1,25802	0,47601	0,14201	0,64000
8	1,25300	0,47000	0,13000	0,65300
9	1,22000	0,48000	0,14000	0,60000
10	1,24000	0,47000	0,16000	0,61000
11	1,30302	0,47401	0,12000	0,70901
12	1,29000	0,46000	0,15000	0,68000
13	1,33100	0,51000	0,14000	0,68100
14	1,30801	0,47601	0,13200	0,70000
15	1,34601	0,45601	0,13000	0,76000
16	1,23000	0,45000	0,12000	0,66000
17	1,26000	0,48000	0,16000	0,62000
18	1,24601	0,50601	0,14000	0,60000
19	1,24000	0,48000	0,15000	0,61000
20	1,29200	0,45200	0,17000	0,67000
21	1,31000	0,48000	0,16000	0,67000
22	1,27900	0,50000	0,15900	0,62000
23	1,34402	0,47302	0,12000	0,75100
24	1,29000	0,46000	0,15000	0,68000
25	1,25401	0,44301	0,15100	0,66000
26	1,27501	0,46901	0,14600	0,66000
27	1,26000	0,45000	0,15000	0,66000
28	1,25000	0,45000	0,15000	0,65000
29	1,25002	0,45201	0,14801	0,65000
30	1,38500	0,48500	0,15000	0,75000
Lower	**1,22000**	**0,44301**	**0,12000**	**0,60000**
Highest	**1,38500**	**0,51000**	**0,17000**	**0,76000**
Average	**1,28067**	**0,47317**	**0,14493**	**0,66257**
Standard Deviation	**0,04618**	**0,01767**	**0,01367**	**0,04717**

Source – The author.

B.3 Writing Transactions

Tables 13, 14, 15, 16 and 17 detail the writing time, considering each one of the thirty repetitions and according to the file size.

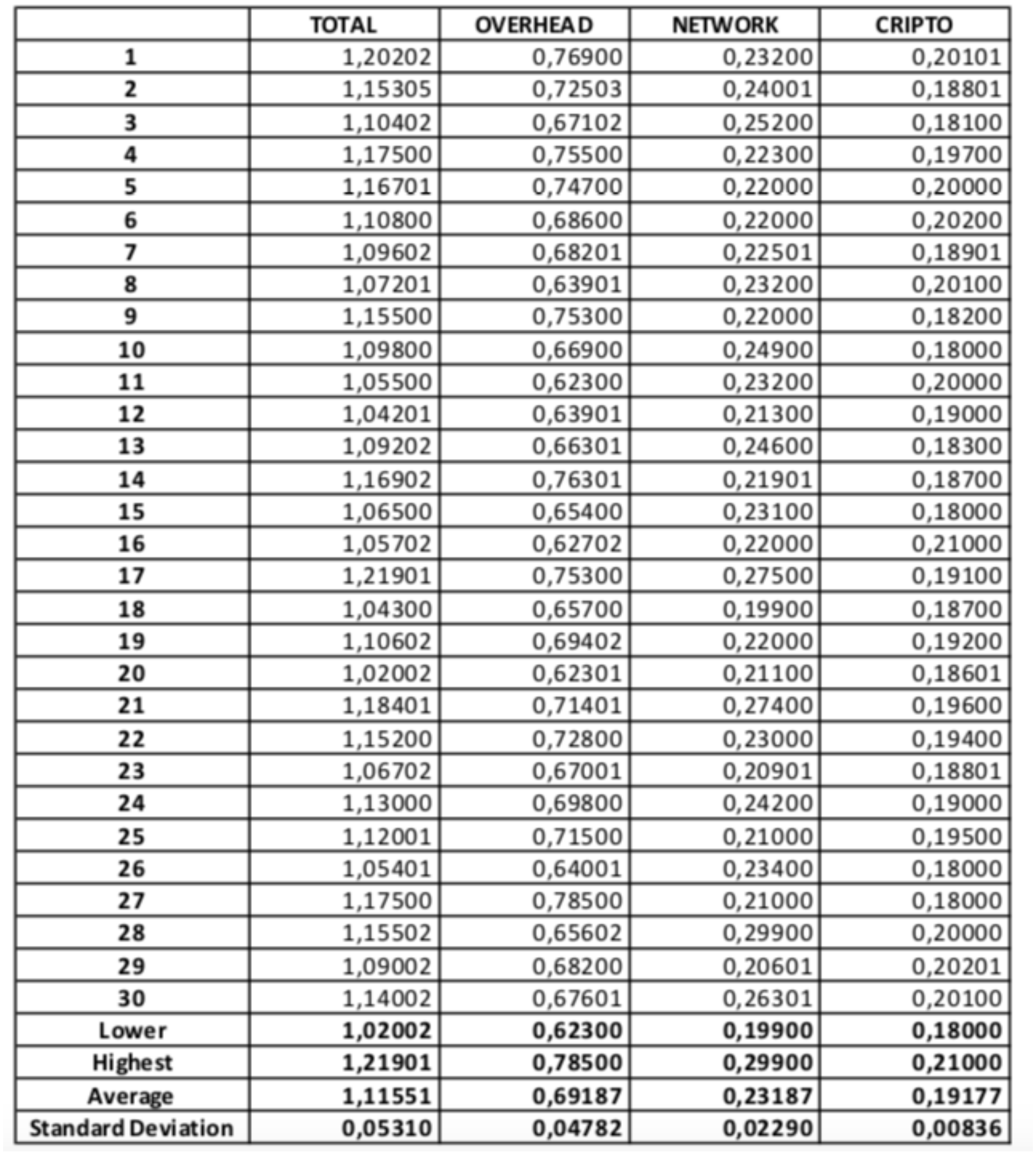
Table 13 – Writing 8KB files



	TOTAL	OVERHEAD	NETWORK	CRIPTO
1	0,97301	0,67901	0,10000	0,19400
2	0,97701	0,68801	0,11400	0,17500
3	1,05401	0,74900	0,11300	0,19200
4	1,00200	0,69700	0,11000	0,19500
5	0,98302	0,70202	0,09900	0,18200
6	1,01001	0,73700	0,11000	0,16300
7	1,04601	0,73001	0,13000	0,18600
8	0,99801	0,69801	0,12000	0,18000
9	0,94501	0,65401	0,11100	0,18000
10	0,99101	0,68101	0,12000	0,19000
11	0,98901	0,68901	0,11000	0,19000
12	0,96700	0,68700	0,12000	0,16000
13	0,99300	0,71100	0,10000	0,18200
14	1,01401	0,71301	0,11100	0,19000
15	0,97800	0,71000	0,10000	0,16800
16	1,01800	0,70500	0,13000	0,18300
17	0,96100	0,67200	0,11600	0,17300
18	1,00200	0,69300	0,11000	0,19900
19	0,98400	0,66900	0,12000	0,19500
20	1,00900	0,69500	0,13000	0,18400
21	1,05101	0,74400	0,12300	0,18400
22	1,01901	0,73400	0,08800	0,19700
23	0,98402	0,69001	0,11300	0,18100
24	0,98800	0,69400	0,11100	0,18300
25	1,01000	0,70000	0,12000	0,19000
26	1,01701	0,71500	0,11000	0,19200
27	1,02000	0,71000	0,12000	0,19000
28	1,03102	0,70801	0,13701	0,18600
29	0,99000	0,69000	0,11000	0,19000
30	1,06201	0,76701	0,10500	0,19000
Lower	**0,94501**	**0,65401**	**0,08800**	**0,16000**
Highest	**1,06201**	**0,76701**	**0,13701**	**0,19900**
Average	**1,00221**	**0,70371**	**0,11370**	**0,18480**
Standard Deviation	**0,02806**	**0,02496**	**0,01065**	**0,00952**

Source – The author.

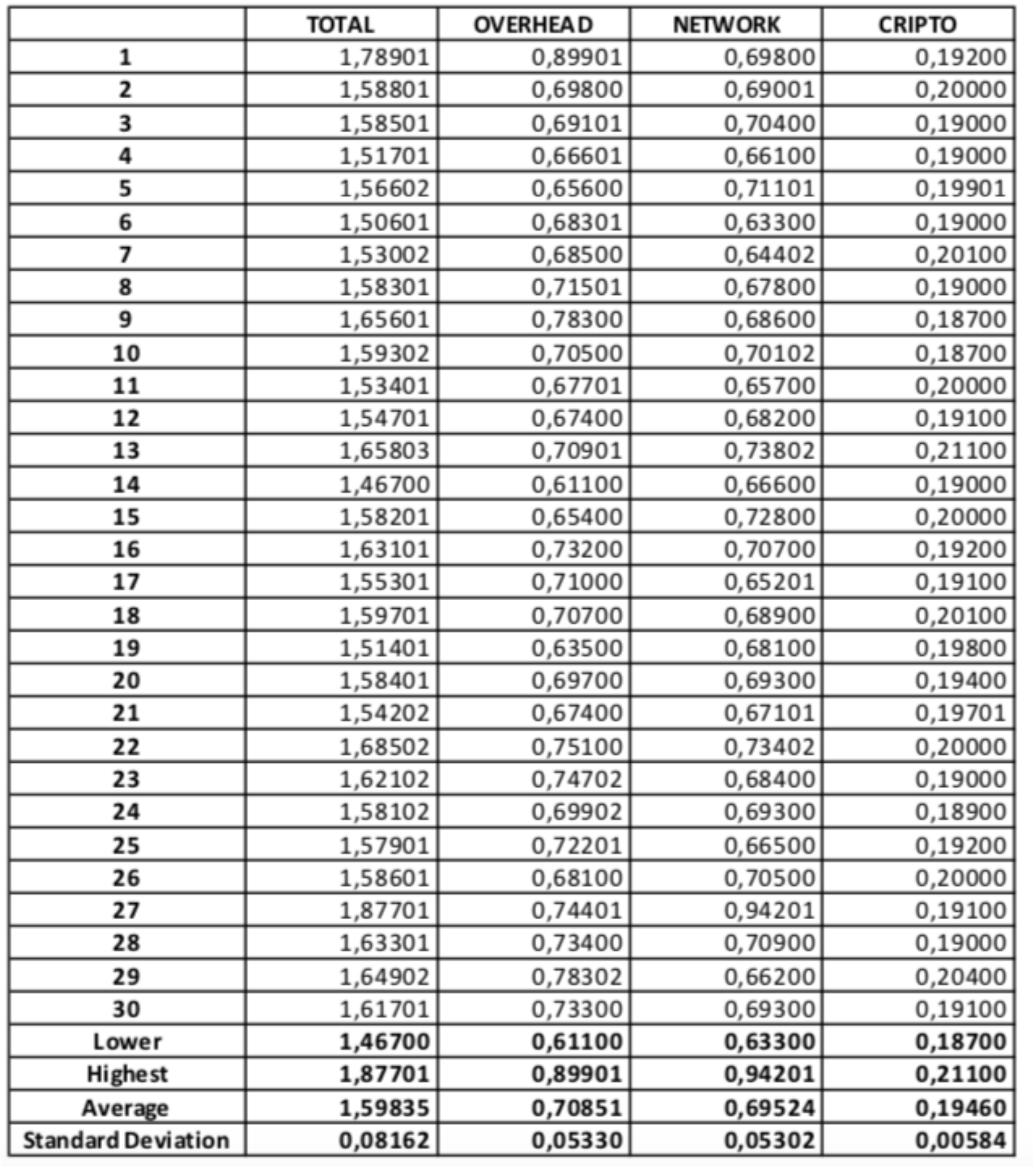
Table 14 – Writing 32KB files



	TOTAL	OVERHEAD	NETWORK	CRIPTO
1	1,20202	0,76900	0,23200	0,20101
2	1,15305	0,72503	0,24001	0,18801
3	1,10402	0,67102	0,25200	0,18100
4	1,17500	0,75500	0,22300	0,19700
5	1,16701	0,74700	0,22000	0,20000
6	1,10800	0,68600	0,22000	0,20200
7	1,09602	0,68201	0,22501	0,18901
8	1,07201	0,63901	0,23200	0,20100
9	1,15500	0,75300	0,22000	0,18200
10	1,09800	0,66900	0,24900	0,18000
11	1,05500	0,62300	0,23200	0,20000
12	1,04201	0,63901	0,21300	0,19000
13	1,09202	0,66301	0,24600	0,18300
14	1,16902	0,76301	0,21901	0,18700
15	1,06500	0,65400	0,23100	0,18000
16	1,05702	0,62702	0,22000	0,21000
17	1,21901	0,75300	0,27500	0,19100
18	1,04300	0,65700	0,19900	0,18700
19	1,10602	0,69402	0,22000	0,19200
20	1,02002	0,62301	0,21100	0,18601
21	1,18401	0,71401	0,27400	0,19600
22	1,15200	0,72800	0,23000	0,19400
23	1,06702	0,67001	0,20901	0,18801
24	1,13000	0,69800	0,24200	0,19000
25	1,12001	0,71500	0,21000	0,19500
26	1,05401	0,64001	0,23400	0,18000
27	1,17500	0,78500	0,21000	0,18000
28	1,15502	0,65602	0,29900	0,20000
29	1,09002	0,68200	0,20601	0,20201
30	1,14002	0,67601	0,26301	0,20100
Lower	**1,02002**	**0,62300**	**0,19900**	**0,18000**
Highest	**1,21901**	**0,78500**	**0,29900**	**0,21000**
Average	**1,11551**	**0,69187**	**0,23187**	**0,19177**
Standard Deviation	**0,05310**	**0,04782**	**0,02290**	**0,00836**

Source – The author.

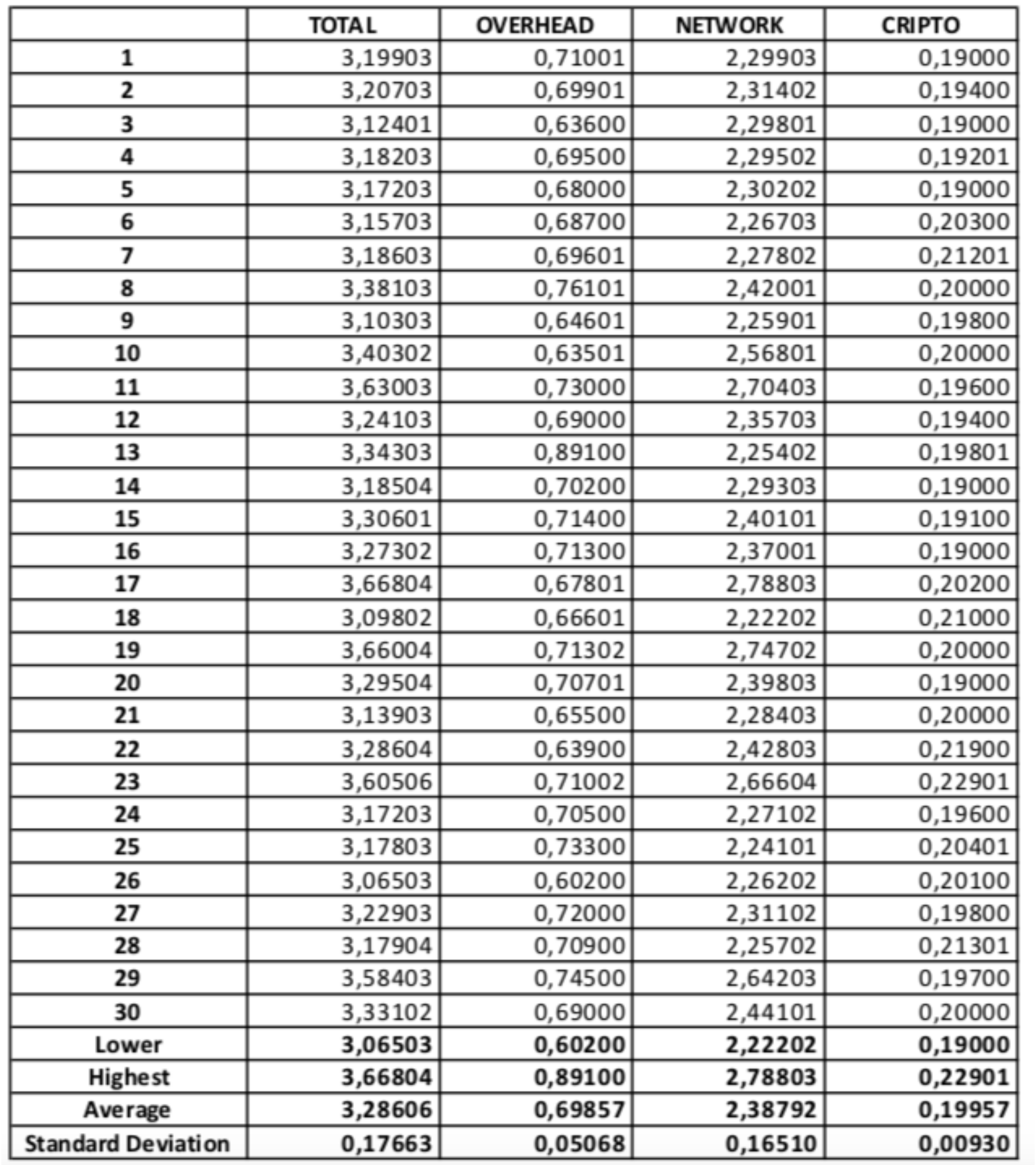
Table 15 – Writing 128KB files



	TOTAL	OVERHEAD	NETWORK	CRIPTO
1	1,78901	0,89901	0,69800	0,19200
2	1,58801	0,69800	0,69001	0,20000
3	1,58501	0,69101	0,70400	0,19000
4	1,51701	0,66601	0,66100	0,19000
5	1,56602	0,65600	0,71101	0,19901
6	1,50601	0,68301	0,63300	0,19000
7	1,53002	0,68500	0,64402	0,20100
8	1,58301	0,71501	0,67800	0,19000
9	1,65601	0,78300	0,68600	0,18700
10	1,59302	0,70500	0,70102	0,18700
11	1,53401	0,67701	0,65700	0,20000
12	1,54701	0,67400	0,68200	0,19100
13	1,65803	0,70901	0,73802	0,21100
14	1,46700	0,61100	0,66600	0,19000
15	1,58201	0,65400	0,72800	0,20000
16	1,63101	0,73200	0,70700	0,19200
17	1,55301	0,71000	0,65201	0,19100
18	1,59701	0,70700	0,68900	0,20100
19	1,51401	0,63500	0,68100	0,19800
20	1,58401	0,69700	0,69300	0,19400
21	1,54202	0,67400	0,67101	0,19701
22	1,68502	0,75100	0,73402	0,20000
23	1,62102	0,74702	0,68400	0,19000
24	1,58102	0,69902	0,69300	0,18900
25	1,57901	0,72201	0,66500	0,19200
26	1,58601	0,68100	0,70500	0,20000
27	1,87701	0,74401	0,94201	0,19100
28	1,63301	0,73400	0,70900	0,19000
29	1,64902	0,78302	0,66200	0,20400
30	1,61701	0,73300	0,69300	0,19100
Lower	**1,46700**	**0,61100**	**0,63300**	**0,18700**
Highest	**1,87701**	**0,89901**	**0,94201**	**0,21100**
Average	**1,59835**	**0,70851**	**0,69524**	**0,19460**
Standard Deviation	**0,08162**	**0,05330**	**0,05302**	**0,00584**

Source – The author.

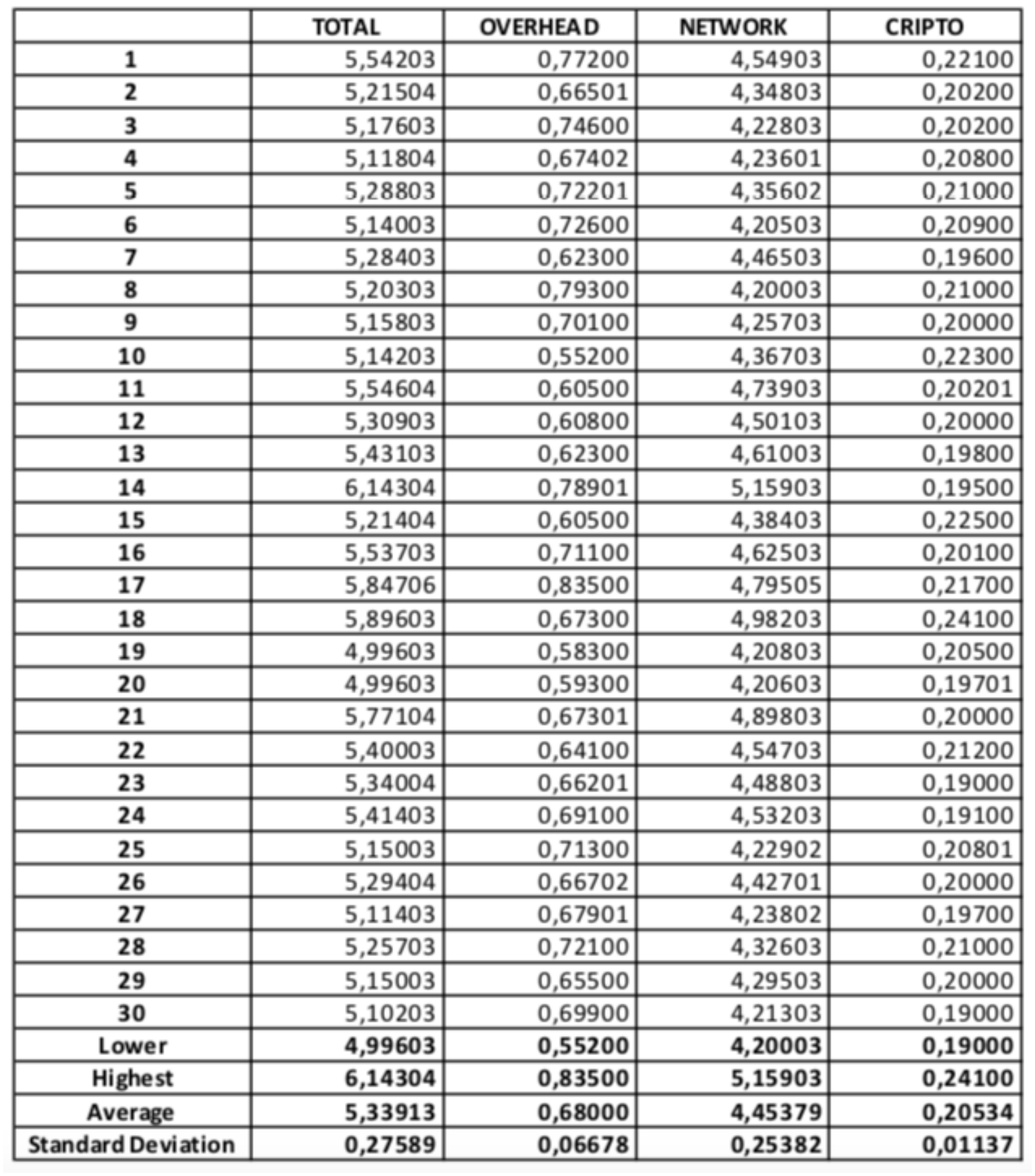
Table 16 – Writing 512KB files



	TOTAL	OVERHEAD	NETWORK	CRIPTO
1	3,19903	0,71001	2,29903	0,19000
2	3,20703	0,69901	2,31402	0,19400
3	3,12401	0,63600	2,29801	0,19000
4	3,18203	0,69500	2,29502	0,19201
5	3,17203	0,68000	2,30202	0,19000
6	3,15703	0,68700	2,26703	0,20300
7	3,18603	0,69601	2,27802	0,21201
8	3,38103	0,76101	2,42001	0,20000
9	3,10303	0,64601	2,25901	0,19800
10	3,40302	0,63501	2,56801	0,20000
11	3,63003	0,73000	2,70403	0,19600
12	3,24103	0,69000	2,35703	0,19400
13	3,34303	0,89100	2,25402	0,19801
14	3,18504	0,70200	2,29303	0,19000
15	3,30601	0,71400	2,40101	0,19100
16	3,27302	0,71300	2,37001	0,19000
17	3,66804	0,67801	2,78803	0,20200
18	3,09802	0,66601	2,22202	0,21000
19	3,66004	0,71302	2,74702	0,20000
20	3,29504	0,70701	2,39803	0,19000
21	3,13903	0,65500	2,28403	0,20000
22	3,28604	0,63900	2,42803	0,21900
23	3,60506	0,71002	2,66604	0,22901
24	3,17203	0,70500	2,27102	0,19600
25	3,17803	0,73300	2,24101	0,20401
26	3,06503	0,60200	2,26202	0,20100
27	3,22903	0,72000	2,31102	0,19800
28	3,17904	0,70900	2,25702	0,21301
29	3,58403	0,74500	2,64203	0,19700
30	3,33102	0,69000	2,44101	0,20000
Lower	**3,06503**	**0,60200**	**2,22202**	**0,19000**
Highest	**3,66804**	**0,89100**	**2,78803**	**0,22901**
Average	**3,28606**	**0,69857**	**2,38792**	**0,19957**
Standard Deviation	**0,17663**	**0,05068**	**0,16510**	**0,00930**

Source – The author.

Table 17 – Writing 1024KB files



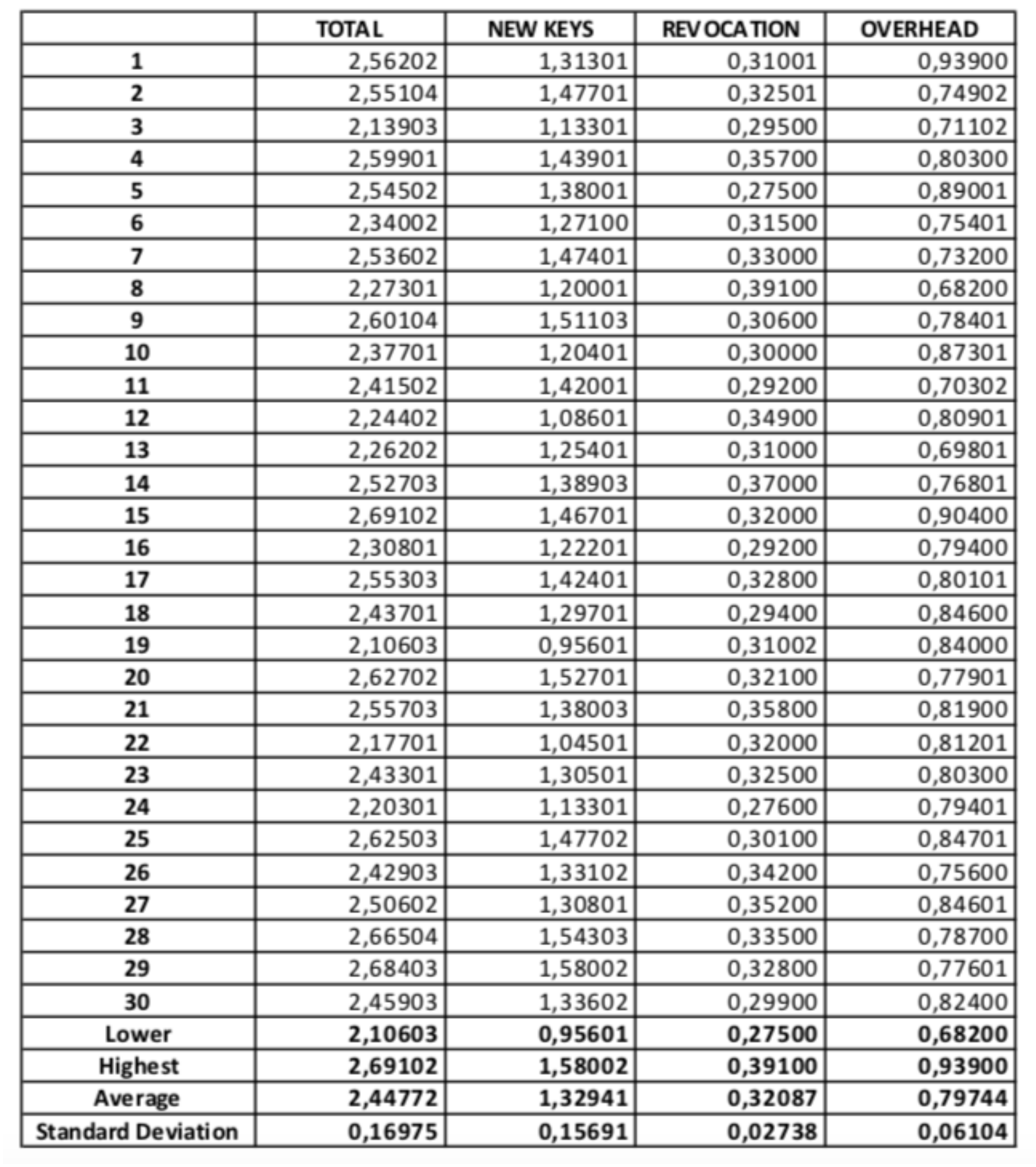
	TOTAL	OVERHEAD	NETWORK	CRIPTO
1	5,54203	0,77200	4,54903	0,22100
2	5,21504	0,66501	4,34803	0,20200
3	5,17603	0,74600	4,22803	0,20200
4	5,11804	0,67402	4,23601	0,20800
5	5,28803	0,72201	4,35602	0,21000
6	5,14003	0,72600	4,20503	0,20900
7	5,28403	0,62300	4,46503	0,19600
8	5,20303	0,79300	4,20003	0,21000
9	5,15803	0,70100	4,25703	0,20000
10	5,14203	0,55200	4,36703	0,22300
11	5,54604	0,60500	4,73903	0,20201
12	5,30903	0,60800	4,50103	0,20000
13	5,43103	0,62300	4,61003	0,19800
14	6,14304	0,78901	5,15903	0,19500
15	5,21404	0,60500	4,38403	0,22500
16	5,53703	0,71100	4,62503	0,20100
17	5,84706	0,83500	4,79505	0,21700
18	5,89603	0,67300	4,98203	0,24100
19	4,99603	0,58300	4,20803	0,20500
20	4,99603	0,59300	4,20603	0,19701
21	5,77104	0,67301	4,89803	0,20000
22	5,40003	0,64100	4,54703	0,21200
23	5,34004	0,66201	4,48803	0,19000
24	5,41403	0,69100	4,53203	0,19100
25	5,15003	0,71300	4,22902	0,20801
26	5,29404	0,66702	4,42701	0,20000
27	5,11403	0,67901	4,23802	0,19700
28	5,25703	0,72100	4,32603	0,21000
29	5,15003	0,65500	4,29503	0,20000
30	5,10203	0,69900	4,21303	0,19000
Lower	**4,99603**	**0,55200**	**4,20003**	**0,19000**
Highest	**6,14304**	**0,83500**	**5,15903**	**0,24100**
Average	**5,33913**	**0,68000**	**4,45379**	**0,20534**
Standard Deviation	**0,27589**	**0,06678**	**0,25382**	**0,01137**

Source – The author.

B.4 Updating Permissions Transactions

Tables 18, 19, 20, 21 and 22 detail the time for updating the permissions, considering each one of the thirty repetitions and according to the file size.

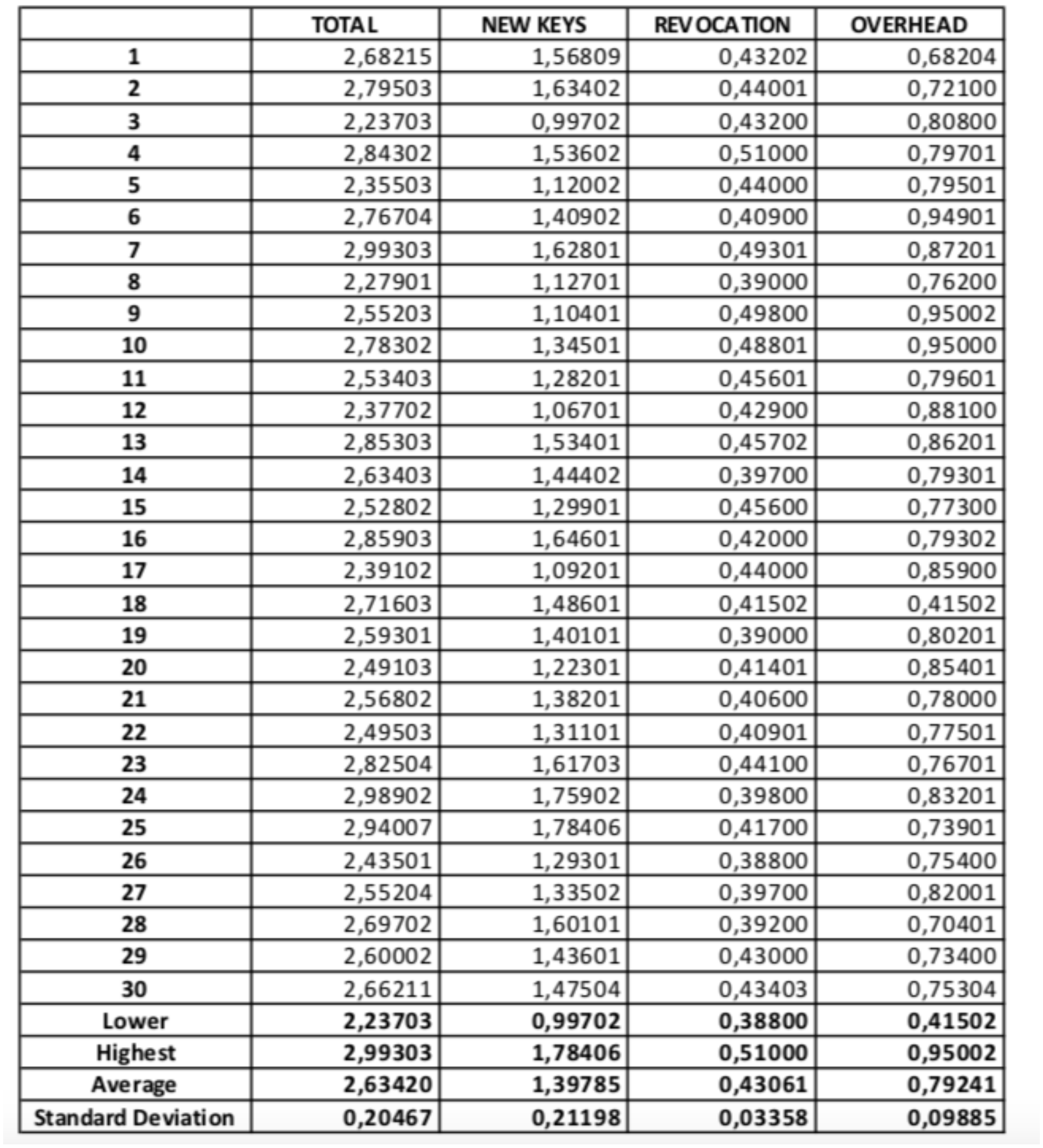
Table 18 – Updating permissions: 8KB files



	TOTAL	NEW KEYS	REVOCATION	OVERHEAD
1	2,56202	1,31301	0,31001	0,93900
2	2,55104	1,47701	0,32501	0,74902
3	2,13903	1,13301	0,29500	0,71102
4	2,59901	1,43901	0,35700	0,80300
5	2,54502	1,38001	0,27500	0,89001
6	2,34002	1,27100	0,31500	0,75401
7	2,53602	1,47401	0,33000	0,73200
8	2,27301	1,20001	0,39100	0,68200
9	2,60104	1,51103	0,30600	0,78401
10	2,37701	1,20401	0,30000	0,87301
11	2,41502	1,42001	0,29200	0,70302
12	2,24402	1,08601	0,34900	0,80901
13	2,26202	1,25401	0,31000	0,69801
14	2,52703	1,38903	0,37000	0,76801
15	2,69102	1,46701	0,32000	0,90400
16	2,30801	1,22201	0,29200	0,79400
17	2,55303	1,42401	0,32800	0,80101
18	2,43701	1,29701	0,29400	0,84600
19	2,10603	0,95601	0,31002	0,84000
20	2,62702	1,52701	0,32100	0,77901
21	2,55703	1,38003	0,35800	0,81900
22	2,17701	1,04501	0,32000	0,81201
23	2,43301	1,30501	0,32500	0,80300
24	2,20301	1,13301	0,27600	0,79401
25	2,62503	1,47702	0,30100	0,84701
26	2,42903	1,33102	0,34200	0,75600
27	2,50602	1,30801	0,35200	0,84601
28	2,66504	1,54303	0,33500	0,78700
29	2,68403	1,58002	0,32800	0,77601
30	2,45903	1,33602	0,29900	0,82400
Lower	**2,10603**	**0,95601**	**0,27500**	**0,68200**
Highest	**2,69102**	**1,58002**	**0,39100**	**0,93900**
Average	**2,44772**	**1,32941**	**0,32087**	**0,79744**
Standard Deviation	**0,16975**	**0,15691**	**0,02738**	**0,06104**

Source – The author.

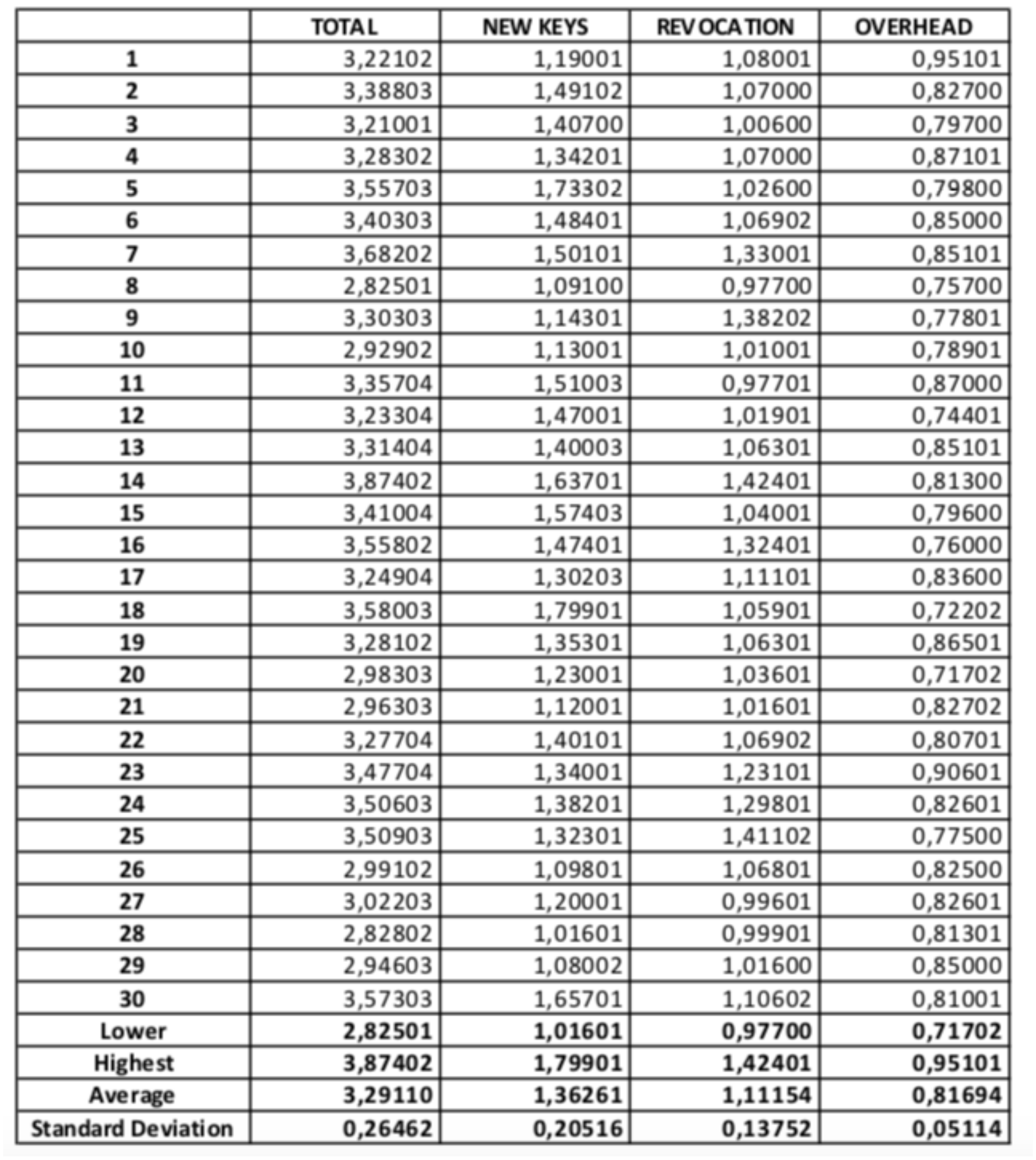
Table 19 – Updating permissions: 32KB files



	TOTAL	NEW KEYS	REVOCATION	OVERHEAD
1	2,68215	1,56809	0,43202	0,68204
2	2,79503	1,63402	0,44001	0,72100
3	2,23703	0,99702	0,43200	0,80800
4	2,84302	1,53602	0,51000	0,79701
5	2,35503	1,12002	0,44000	0,79501
6	2,76704	1,40902	0,40900	0,94901
7	2,99303	1,62801	0,49301	0,87201
8	2,27901	1,12701	0,39000	0,76200
9	2,55203	1,10401	0,49800	0,95002
10	2,78302	1,34501	0,48801	0,95000
11	2,53403	1,28201	0,45601	0,79601
12	2,37702	1,06701	0,42900	0,88100
13	2,85303	1,53401	0,45702	0,86201
14	2,63403	1,44402	0,39700	0,79301
15	2,52802	1,29901	0,45600	0,77300
16	2,85903	1,64601	0,42000	0,79302
17	2,39102	1,09201	0,44000	0,85900
18	2,71603	1,48601	0,41502	0,41502
19	2,59301	1,40101	0,39000	0,80201
20	2,49103	1,22301	0,41401	0,85401
21	2,56802	1,38201	0,40600	0,78000
22	2,49503	1,31101	0,40901	0,77501
23	2,82504	1,61703	0,44100	0,76701
24	2,98902	1,75902	0,39800	0,83201
25	2,94007	1,78406	0,41700	0,73901
26	2,43501	1,29301	0,38800	0,75400
27	2,55204	1,33502	0,39700	0,82001
28	2,69702	1,60101	0,39200	0,70401
29	2,60002	1,43601	0,43000	0,73400
30	2,66211	1,47504	0,43403	0,75304
Lower	**2,23703**	**0,99702**	**0,38800**	**0,41502**
Highest	**2,99303**	**1,78406**	**0,51000**	**0,95002**
Average	**2,63420**	**1,39785**	**0,43061**	**0,79241**
Standard Deviation	**0,20467**	**0,21198**	**0,03358**	**0,09885**

Source – The author.

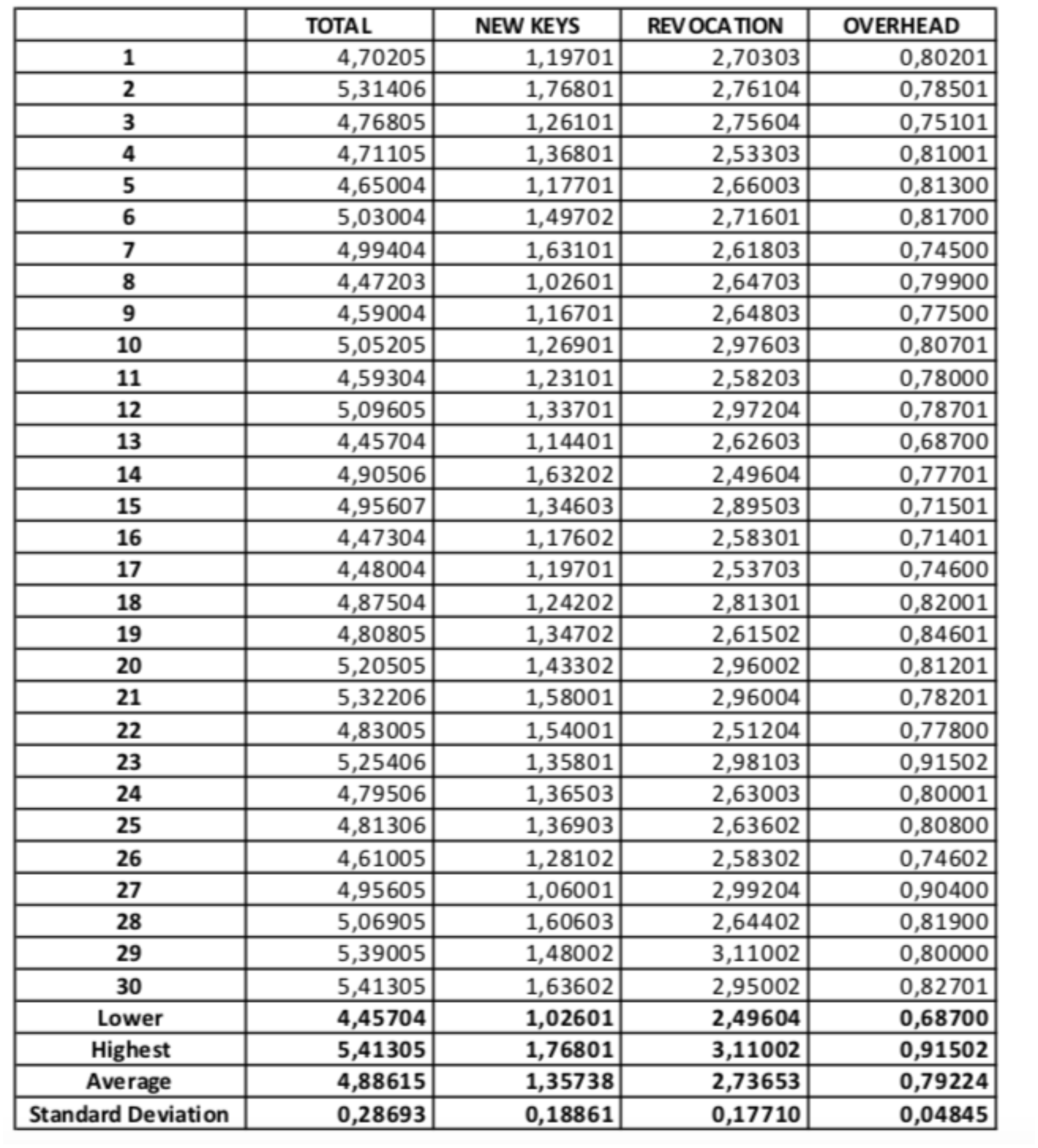
Table 20 – Updating permissions: 128KB files



	TOTAL	NEW KEYS	REVOCATION	OVERHEAD
1	3,22102	1,19001	1,08001	0,95101
2	3,38803	1,49102	1,07000	0,82700
3	3,21001	1,40700	1,00600	0,79700
4	3,28302	1,34201	1,07000	0,87101
5	3,55703	1,73302	1,02600	0,79800
6	3,40303	1,48401	1,06902	0,85000
7	3,68202	1,50101	1,33001	0,85101
8	2,82501	1,09100	0,97700	0,75700
9	3,30303	1,14301	1,38202	0,77801
10	2,92902	1,13001	1,01001	0,78901
11	3,35704	1,51003	0,97701	0,87000
12	3,23304	1,47001	1,01901	0,74401
13	3,31404	1,40003	1,06301	0,85101
14	3,87402	1,63701	1,42401	0,81300
15	3,41004	1,57403	1,04001	0,79600
16	3,55802	1,47401	1,32401	0,76000
17	3,24904	1,30203	1,11101	0,83600
18	3,58003	1,79901	1,05901	0,72202
19	3,28102	1,35301	1,06301	0,86501
20	2,98303	1,23001	1,03601	0,71702
21	2,96303	1,12001	1,01601	0,82702
22	3,27704	1,40101	1,06902	0,80701
23	3,47704	1,34001	1,23101	0,90601
24	3,50603	1,38201	1,29801	0,82601
25	3,50903	1,32301	1,41102	0,77500
26	2,99102	1,09801	1,06801	0,82500
27	3,02203	1,20001	0,99601	0,82601
28	2,82802	1,01601	0,99901	0,81301
29	2,94603	1,08002	1,01600	0,85000
30	3,57303	1,65701	1,10602	0,81001
Lower	**2,82501**	**1,01601**	**0,97700**	**0,71702**
Highest	**3,87402**	**1,79901**	**1,42401**	**0,95101**
Average	**3,29110**	**1,36261**	**1,11154**	**0,81694**
Standard Deviation	**0,26462**	**0,20516**	**0,13752**	**0,05114**

Source – The author.

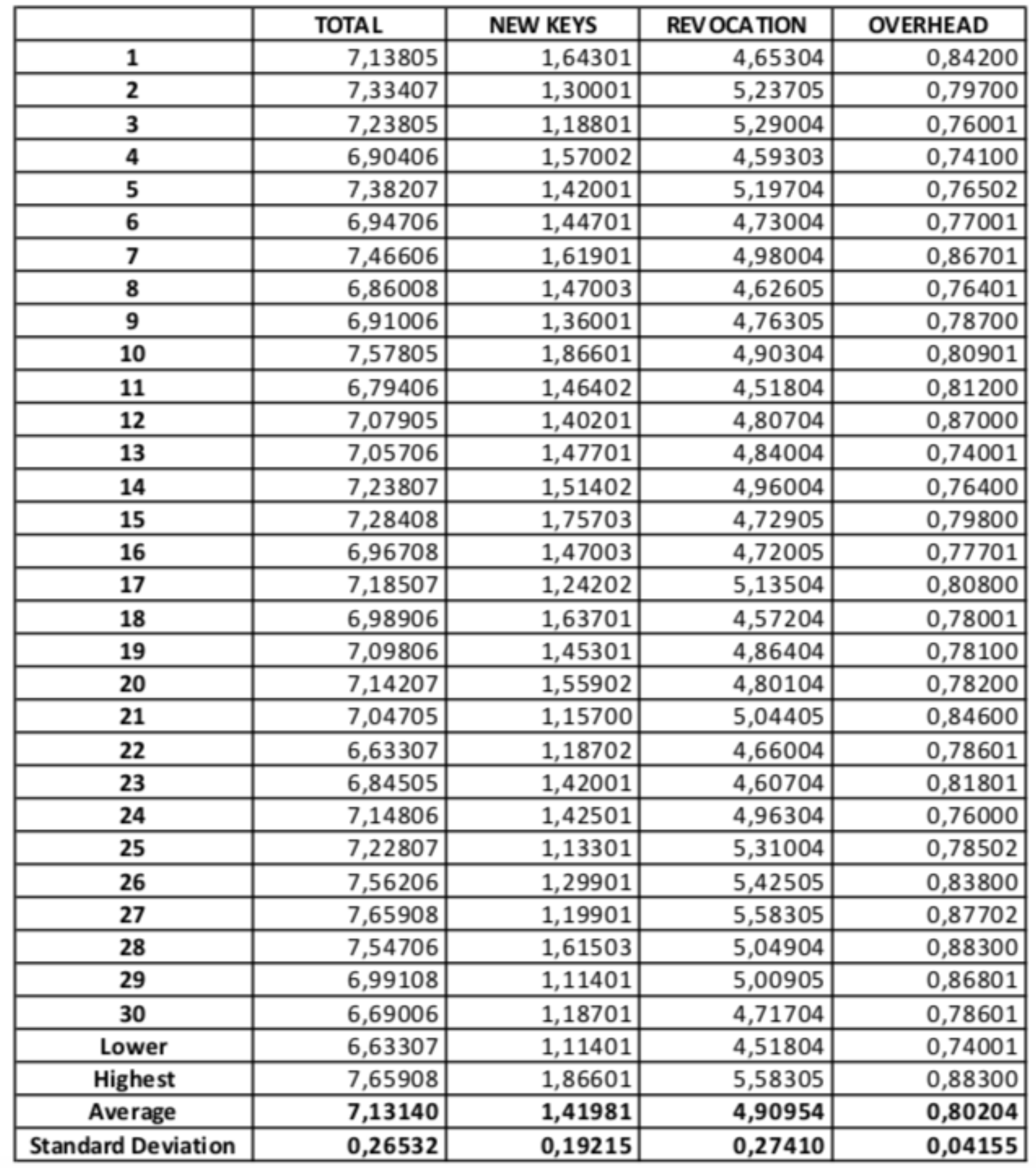
Table 21 – Updating permissions: 512KB files



	TOTAL	NEW KEYS	REVOCATION	OVERHEAD
1	4,70205	1,19701	2,70303	0,80201
2	5,31406	1,76801	2,76104	0,78501
3	4,76805	1,26101	2,75604	0,75101
4	4,71105	1,36801	2,53303	0,81001
5	4,65004	1,17701	2,66003	0,81300
6	5,03004	1,49702	2,71601	0,81700
7	4,99404	1,63101	2,61803	0,74500
8	4,47203	1,02601	2,64703	0,79900
9	4,59004	1,16701	2,64803	0,77500
10	5,05205	1,26901	2,97603	0,80701
11	4,59304	1,23101	2,58203	0,78000
12	5,09605	1,33701	2,97204	0,78701
13	4,45704	1,14401	2,62603	0,68700
14	4,90506	1,63202	2,49604	0,77701
15	4,95607	1,34603	2,89503	0,71501
16	4,47304	1,17602	2,58301	0,71401
17	4,48004	1,19701	2,53703	0,74600
18	4,87504	1,24202	2,81301	0,82001
19	4,80805	1,34702	2,61502	0,84601
20	5,20505	1,43302	2,96002	0,81201
21	5,32206	1,58001	2,96004	0,78201
22	4,83005	1,54001	2,51204	0,77800
23	5,25406	1,35801	2,98103	0,91502
24	4,79506	1,36503	2,63003	0,80001
25	4,81306	1,36903	2,63602	0,80800
26	4,61005	1,28102	2,58302	0,74602
27	4,95605	1,06001	2,99204	0,90400
28	5,06905	1,60603	2,64402	0,81900
29	5,39005	1,48002	3,11002	0,80000
30	5,41305	1,63602	2,95002	0,82701
Lower	**4,45704**	**1,02601**	**2,49604**	**0,68700**
Highest	**5,41305**	**1,76801**	**3,11002**	**0,91502**
Average	**4,88615**	**1,35738**	**2,73653**	**0,79224**
Standard Deviation	**0,28693**	**0,18861**	**0,17710**	**0,04845**

Source – The author.

Table 22 – Updating permissions: 1024KB files



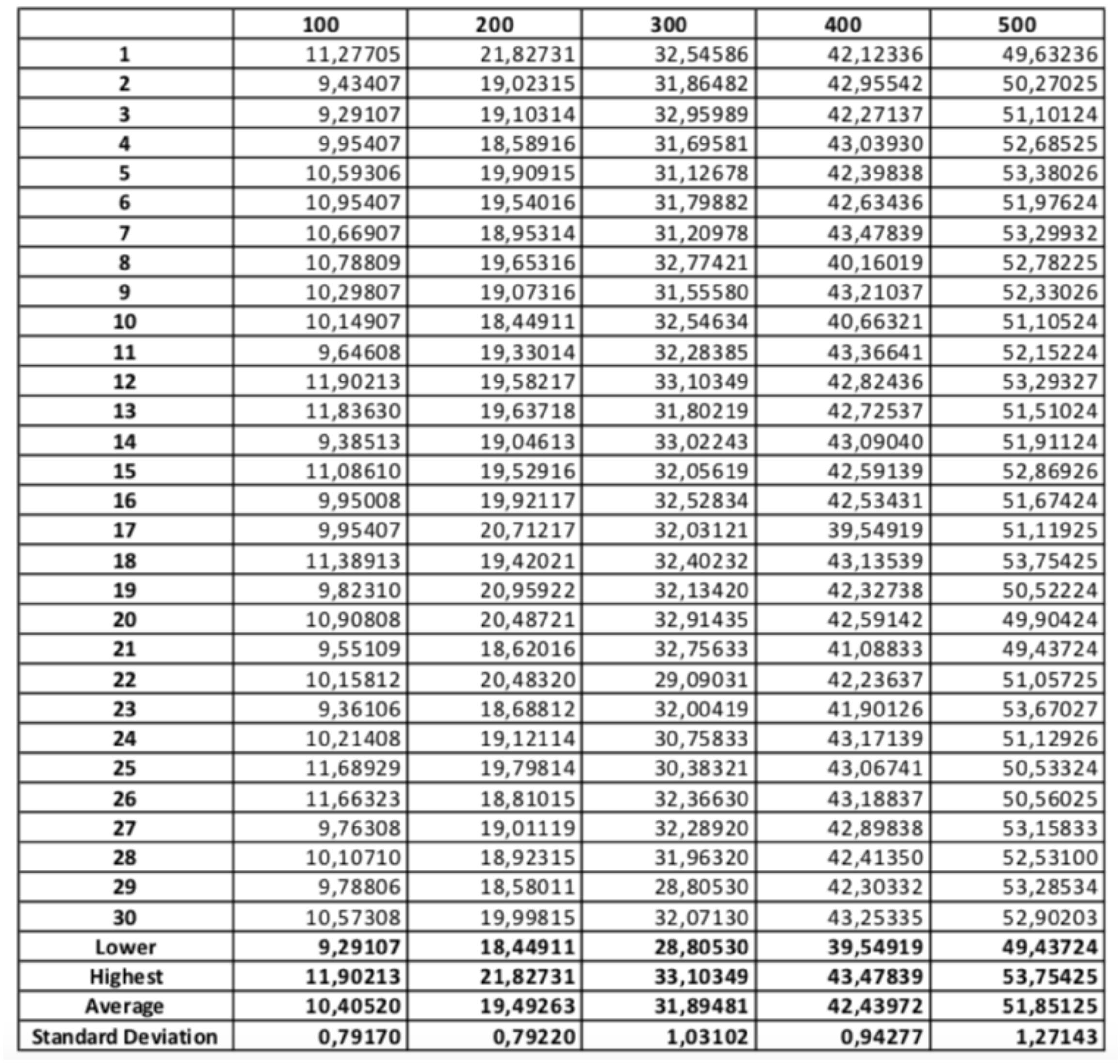
	TOTAL	NEW KEYS	REVOCATION	OVERHEAD
1	7,13805	1,64301	4,65304	0,84200
2	7,33407	1,30001	5,23705	0,79700
3	7,23805	1,18801	5,29004	0,76001
4	6,90406	1,57002	4,59303	0,74100
5	7,38207	1,42001	5,19704	0,76502
6	6,94706	1,44701	4,73004	0,77001
7	7,46606	1,61901	4,98004	0,86701
8	6,86008	1,47003	4,62605	0,76401
9	6,91006	1,36001	4,76305	0,78700
10	7,57805	1,86601	4,90304	0,80901
11	6,79406	1,46402	4,51804	0,81200
12	7,07905	1,40201	4,80704	0,87000
13	7,05706	1,47701	4,84004	0,74001
14	7,23807	1,51402	4,96004	0,76400
15	7,28408	1,75703	4,72905	0,79800
16	6,96708	1,47003	4,72005	0,77701
17	7,18507	1,24202	5,13504	0,80800
18	6,98906	1,63701	4,57204	0,78001
19	7,09806	1,45301	4,86404	0,78100
20	7,14207	1,55902	4,80104	0,78200
21	7,04705	1,15700	5,04405	0,84600
22	6,63307	1,18702	4,66004	0,78601
23	6,84505	1,42001	4,60704	0,81801
24	7,14806	1,42501	4,96304	0,76000
25	7,22807	1,13301	5,31004	0,78502
26	7,56206	1,29901	5,42505	0,83800
27	7,65908	1,19901	5,58305	0,87702
28	7,54706	1,61503	5,04904	0,88300
29	6,99108	1,11401	5,00905	0,86801
30	6,69006	1,18701	4,71704	0,78601
Lower	6,63307	1,11401	4,51804	0,74001
Highest	7,65908	1,86601	5,58305	0,88300
Average	**7,13140**	**1,41981**	**4,90954**	**0,80204**
Standard Deviation	**0,26532**	**0,19215**	**0,27410**	**0,04155**

Source – The author.

B.5 Auditing

The experiments referring to the auditing were performed using different sizes of the log to demonstrate that the time for auditing increases linearly according to this size. Table 23 displays the time of each repetition of these experiments.

Table 23 – Auditing time and log size



	100	200	300	400	500
1	11,27705	21,82731	32,54586	42,12336	49,63236
2	9,43407	19,02315	31,86482	42,95542	50,27025
3	9,29107	19,10314	32,95989	42,27137	51,10124
4	9,95407	18,58916	31,69581	43,03930	52,68525
5	10,59306	19,90915	31,12678	42,39838	53,38026
6	10,95407	19,54016	31,79882	42,63436	51,97624
7	10,66907	18,95314	31,20978	43,47839	53,29932
8	10,78809	19,65316	32,77421	40,16019	52,78225
9	10,29807	19,07316	31,55580	43,21037	52,33026
10	10,14907	18,44911	32,54634	40,66321	51,10524
11	9,64608	19,33014	32,28385	43,36641	52,15224
12	11,90213	19,58217	33,10349	42,82436	53,29327
13	11,83630	19,63718	31,80219	42,72537	51,51024
14	9,38513	19,04613	33,02243	43,09040	51,91124
15	11,08610	19,52916	32,05619	42,59139	52,86926
16	9,95008	19,92117	32,52834	42,53431	51,67424
17	9,95407	20,71217	32,03121	39,54919	51,11925
18	11,38913	19,42021	32,40232	43,13539	53,75425
19	9,82310	20,95922	32,13420	42,32738	50,52224
20	10,90808	20,48721	32,91435	42,59142	49,90424
21	9,55109	18,62016	32,75633	41,08833	49,43724
22	10,15812	20,48320	29,09031	42,23637	51,05725
23	9,36106	18,68812	32,00419	41,90126	53,67027
24	10,21408	19,12114	30,75833	43,17139	51,12926
25	11,68929	19,79814	30,38321	43,06741	50,53324
26	11,66323	18,81015	32,36630	43,18837	50,56025
27	9,76308	19,01119	32,28920	42,89838	53,15833
28	10,10710	18,92315	31,96320	42,41350	52,53100
29	9,78806	18,58011	28,80530	42,30332	53,28534
30	10,57308	19,99815	32,07130	43,25335	52,90203
Lower	**9,29107**	**18,44911**	**28,80530**	**39,54919**	**49,43724**
Highest	**11,90213**	**21,82731**	**33,10349**	**43,47839**	**53,75425**
Average	**10,40520**	**19,49263**	**31,89481**	**42,43972**	**51,85125**
Standard Deviation	**0,79170**	**0,79220**	**1,03102**	**0,94277**	**1,27143**

Source – The author.