Centro de Investigación Científica y de Educación Superior de Ensenada, Baja California

---

Doctorado en Ciencias en Ciencias de la Computación

Resilient configurable scheme for data cloud storage

Tesis
para cubrir parcialmente los requisitos necesarios para obtener el grado de
Doctor en Ciencias

Presenta:

Vanessa Miranda López

Ensenada, Baja California, México
2021

Tesis defendida por

Vanessa Miranda López

y aprobada por el siguiente Comité

Dr. Andrey Chernykh
Director de tesis

Miembros del comité
Dr. Carlos Alberto Brizuela Rodríguez

Dr. Edgar Leonel Chávez González

Dr. Mikhail Babenko

Dr. Israel Marck Martínez Pérez
Coordinador del Posgrado en Ciencias de la Computación

Dr. Pedro Negrete Regagnon
Director de Estudios de Posgrado

Resumen de la tesis que presenta **Vanessa Miranda López** como requisito parcial para la obtención del grado de Doctor en Ciencias en Ciencias de la Computación.

Esquema configurable resistente para el almacenamiento de datos en la nube

Resumen aprobado por:

____

Dr. Andrey Chernykh

Director de tesis

El Sistema Numérico del Residuo (RNS) es un sistema no ponderado basado en el Teorema Chino del Residuo (CRT). En RNS, la información se divide en segmentos más pequeños permitiendo operaciones paralelas independientes entre sí. Esta propiedad es una de las principales razones para considerar RNS como una solución en aplicaciones donde se desea optimizar velocidades de procesamiento, bajo consumo de energía y mantener la privacidad e integridad de la información. El concepto importante es la posibilidad de recuperar los datos originales utilizando menos residuos creados inicialmente para mejorar la confiabilidad. Una posible área de aplicación de RNS es el almacenamiento como servicio (STaaS). STaaS es uno de los modelos más prácticos que se ofrecen para el almacenamiento en línea. Si bien, actualmente este servicio puede brindar flexibilidad, escalabilidad e incluso bajos costos de inversión, presenta problemas y desafíos únicos de seguridad y confiabilidad. Por lo tanto, en esta tesis, se presenta el uso de RNS en el diseño de un mecanismo configurable que optimiza la detección y corrección de errores para mejorar la confiabilidad en sistemas que empleen RNS. Nuestra solución se basa en esquemas configurables de Compartición de Secretos, Sistema Numérico de Residuos Redundantes (RRNS), Sistema Numérico de Base Mixta (MRS) y códigos de detección y corrección de errores. El modelo propuesto, llamado 2Lbp-RRNS, utiliza un esquema RRNS de dos niveles y mecanismos de propagación hacia atrás y distancia de Hamming para minimizar los riesgos de confidencialidad, integridad, disponibilidad y pérdida de información asociados con hardware defectuoso, software defectuoso, fallas técnicas, ataques DDoS y modificación no autorizada. Validamos la calidad del modelo mediante un análisis teórico, obteniendo las cotas superiores para la estimación del número de errores detectables y corregibles de nuestra propuesta. De esta manera, demostramos que 2Lbp-RRNS aumenta el número de errores detectados y corregidos en comparación con los modelos presentados en la literatura. Finalmente, como caso de estudio, realizamos una extensa evaluación experimental mediante la simulación de un sistema de almacenamiento de datos utilizando características y propiedades de proveedores de nube reales bajo diferentes escenarios. Para el análisis implementamos los siguientes esquemas de compartición de secretos: dos esquemas clásicos basados en CRT: Mignotte y Asmuth-Bloom, el enfoque AR-RRNS basado en el rango aproximado de RRNS, el código Reed-Solomon, y MRC-RRNS basado en Conversión de Base Mixta (MRC).

Palabras clave: Sistema Numérico del Residuo, Seguridad de datos, Confiabilidad, Corrección de Errores, Incertidumbre

Abstract of the thesis presented by **Vanessa Miranda López** as a partial requirement to obtain the Doctor of Science degree in Computer Science.

Resilient configurable scheme for data cloud storage

Abstract approved by:

____
Dr. Andrey Chernykh
Thesis Director

The Residue Number System (RNS) is a non-weighted system based on the Chinese Remainder Theorem (CRT). In RNS, the information is divided into smaller segments allowing parallel operations independent of each other. This property is one of the main reasons to consider RNS as a solution in applications where it is desired to optimize processing speeds, power consumption, and maintain information's privacy and integrity. The important concept is the possibility of recovering the original data using fewer residues created initially to improve reliability. A possible RNS application area is Storage as a Service (STaaS). STaaS is one of the most practical models offered for online storage. While this service can currently provide flexibility, scalability, and even low investment costs, it presents unique security and reliability issues and challenges. Therefore, in this thesis, RNS is used in designing a configurable mechanism that optimizes the detection and correction of errors to improve the reliability of the system. Our solution is based on configurable Secret Sharing schemes, Redundant Residue Number System (RRNS), Mixed-Radix Number System (MRS), and error detection and correction codes. The proposed model, called 2Lbp-RRNS, uses a two-level RRNS scheme, Hamming distance, and backpropagation mechanisms to minimize the risks of confidentiality, integrity, availability, and information loss associate with faulty hardware, faulty software, technical malfunctions, DDoS attacks, and unauthorized modification. We validate the model's quality through theoretical analysis, obtaining the upper bounds for estimating the number of detectable and correctable errors of our proposal. This way, we demonstrate that 2Lbp-RRNS increases the number of detected and corrected errors compared to the literature models. Finally, as a case study, we carried out an extensive experimental evaluation by simulating a data storage system using real cloud providers' characteristics and properties under different scenarios. For the analysis, we implemented the following secret sharing schemes: two classic CRT-based schemes: Mignotte and Asmuth-Bloom, the AR-RRNS approach based on the approximate range of RRNS, the Reed-Solomon code, and MRC-RRNS based on Mixed Base Conversion (MRC).

Keywords: Residue Number System, Data Security, Reliability, Error Correction, Uncertainty

Dedicatory

To my daughter Allison ….

Acknowledgments

Firstly, I would like to express my sincere gratitude to Dr. Andrei Tchernykh for being a great mentor during this long journey, for your excellent work ethic, but mostly for being an exceptional person. Thank you for deeply contribute to this thesis with hard work through these four years. This work is also yours.

I would like to thank my thesis committee: Dr. Mikhail Babenko, Dr. Carlos Brizuela Rodríguez, and Dr. Edgar Chávez González for their wise advice and insightful comments throughout the thesis.

I acknowledge my gratitude to Fermin Armenta for supporting in my lowest moments. Additionally, I would like to thank my great lab colleagues Bernardo Pulido, Rewer Canosa, and Luis García, for your help and encouragement during the thesis.

Finally, I thank all the people who make research in Mexico possible. Thanks to all the people of Mexico who, through the Consejo Nacional de Ciencia y Tecnología (CONACyT), awarded me a scholarship no. 269338 that allows me to carry out this research work.

Basic notation

D	Original data
Size(D)	Size of the original data D
S	Secret data
$S_i = \|S\|_{p_i}$	Remainder of the division S by module p_i
Z_P	Set $\{0,1,\dots,p-1\}$ for $p \geq 1$, where p is prime
u_j	Upload speed of the $j-th$ cloud provider
d_j	Download speed of the $j-th$ cloud provider
T_D	Total decoding time
T_E	Total encoding time
V_D	Decoding speed
V_E	Encoding speed
T_{up}	Upload time of encrypted data to the cloud provider
T_{dow}	Download time of encrypted data from the cloud provider
V_s	Storing speed
V_{ex}	Extraction speed
T_s	Storing time
T_{ex}	Extraction time
R	Redundancy

One-Level model

$n = \sum_{\mathrm{i}=1}^{\mathrm{m}} n_{\mathrm{i}}$	Number of RRNS moduli
$k \leq n$	Threshold value for secret sharing scheme
(n,k)	RRNS access structure
$r = n-k$	Number of control (redundant) RRNS moduli
p_i	$i-th$ RRNS modulus
$P = \prod_{i=1}^{k} p_i$	Dynamic range of RRNS
$n_i \geq 1$	Number of shares stored in the $i-th$ Cloud
$P_r(k,n)$	Probability of information loss
err_j	Probability of failure of the $j-th$ Cloud

Two-Level model

$\bar{S} \xleftarrow{RRNS} S + E$	Representation of S with error
$\tilde{S}$	Representation of S with error in 2L-RRNS
$\bar{I}$	Tuple of residues with an error
I_E	Subset $\{1,..,n_1\}$power is equal to $\left\lfloor\frac{k_1+n_1}{2}\right\rfloor$
I_D	Subset $\{1,..,n_1\}$power is equal to k_1
$\bar{I}_i$	Tuple of residues with an error
N_D^{2L}	Number of detected errors in 2L-RRNS
N_E^{2L}	Number of corrected errors in 2L-RRNS
N_D^{2Lbp}	Number of detected errors in 2Lbp-RRNS
N_E^{2Lbp}	Number of corrected errors in 2Lbp-RRNS
N_{Dl}	Number of detected errors with knowing error localization
N_{El}	Number of corrected errors with knowing error localization
	First Level
n_1	Number of moduli on the first level
$k_1 \le n_1$	Threshold value on the first level for the secret sharing scheme
$r_1 = n_1 - k_1$	Number of control (redundant) RRNS moduli
$p_{1,i}$	i-th RNS moduli on the first level
$P = \prod_{i=1}^{k_1} p_{1,i}$	Legal dynamic range RRNS on the first level and $S \in [0, P)$
$\bar{P} = \prod_{i=1}^{n_1} p_{1,i}$	$[0, \bar{P} - 1]$ full range
$S_i = \lvert S\rvert_{p_{1,i}}$	Remainder of the division S by modulo $p_{1,i}$
	Second level
$n_{2,i}$	Number of moduli used to calculate S_i for each $i = \overline{1, n_1}$,
$k_{2,i} \le n_{2,i}$	Threshold value for the secret sharing scheme used for S_i,
$r_{2,i} = n_{2,i} - k_{2,i}$	Number of control (redundant) RRNS moduli for S_i,
$p_{2,i,j}$	RRNS modulus used in the representation S_i for each $i = \overline{1, n_1}$ and j= $\overline{1, n_{2,i}}$,
$M_i = \prod_{j=1}^{k_{2,i}} p_{2,i,j} \ge p_{1,i}$	Dynamic range of RRNS definite moduli set and $S_i \in [0, p_{1,i})$ for each $i = \overline{1, n_1}$,
$S_{i,j} = \lvert S_i\rvert_{p_{2,i,j}}$	Remainder of the division S_i by modulo $p_{2,i,j}$ for each $i = \overline{1, n_1}$ and j $= \overline{1, n_{2,i}}$.

Table of contents

List of figures

List of tables

Chapter 1. Introduction

Cloud computing provides many benefits in terms of low costs, data accessibility, archiving, backuping, data sharing, synchronizing multiple devices, and efficient performances. Users/clients can choose resources that fulfill their needs from available Cloud provider pools of multiple Cloud services. This type of provisioning according to the users' demands gives the illusion of infinite resources.

One of these services is Storage as a Service (STaaS) (Kulkarni et al., 2012; Zhao et al., 2009) like Dropbox, GoogleDrive, Mozy, Box, and MEGA. They provide data outsourcing and data sharing with users all around the world. Recently, its popularity has increased considerably for individual users and organizations, eliminating local user storage infrastructures. However, storing data/information in Cloud Storage (CS) raises security issues. The concerns are mainly motivated by security and privacy-related features that require users to fully trust declared security services practices like authentication, confidentiality, integrity, and trust management. There is no guarantee of data privacy since Cloud employees can access all the stored information, and the Cloud storage could be a victim of cyber-attacks (Irwin, 2020). A trustworthy cloud environment is a prerequisite to winning a user's confidence to adopt such technology (Mishra et al., 2018).

One of the approaches to solve these problems is the Residue Number System (RNS) (Garner, 1959), a number system in which integers are represented by their values modulo several pairwise coprime integers called the moduli.

Multi-modular arithmetic of RNS is widely used for computation with large integers. It supports parallel, carry-free addition, borrow-free subtraction, and single-step multiplication without unfinished products. RNS is very useful for certain applications, for instance, Digital Signal Processing (speech and image processing) providing high-speed computations (Bajard et al., 2011; Jenkins, 1978; Jenkins & Leon, 1977; Johnson et al., 1986; Jyothi et al., 2020; Pontarelli et al., 2008).

RNS is speeding up cryptosystems such as Rivest, Shamir, and Adleman (RSA) (Blakley & Borosh, 1979; Shieh et al., 2008; Wu et al., 2001), Elliptic Curve Cryptography (ECC) (Schinianakis et al., 2009), etc.

RNS is also used for efficient implementation of Fully homomorphic encryption schemes (FHE) (Cheon et al., 2018, 2019; Gentry, 2009; Phong et al., 2018).

In communications engineering, the features of RNS are used for reducing time delay, hardware costs, energy consumption of sensor nodes, and enhancing wireless sensor networks reliability (Campobello et al., 2012; Chessa et al., 2004; Jia & Wang, 2013; Junior et al., 2011; Nazarov et al., 2018).

One of the latest research directions of the RNS application is distributed cloud storages. It helps to deal with the unexpected terminations of services, data breaches, technical failures, etc. Celesti et al., (2016) presented the first work that introduces RNS to overcome security issues on Cloud storage. The main idea is to improve long-term reliability and privacy by considering different cloud storage providers by dividing the file into residue segments. Several solutions are proposed based on this concept, such as Galletta et al., (2020); Hema & Durga, (2014); Kar et al., (2016); Miranda-López et al., (2018); A. Tchernykh et al., (2018); A. Tchernykh et al., (2019).

1.1 Related work

Cloud services can be considered secure if they include authentication, data encryption, data recovery, and user protection. Many solutions propose mechanisms and schemes to solve Cloud security issues in distributed environments. In this section, we discuss recent approaches and related works focused on data security and reliability.

1.1.1 Distributed storage systems

To construct a distributed system for data storage and processing, researchers use a variety of approaches. They consider Cloud and grid computing paradigms to develop distributed data storages (Vouk, 2008). These infrastructures have common characteristics but also significant differences. To cope with availability, reliability, risks of data loss, and vendor lock-in, they use Information Dispersal (Rabin, 1990) and Data Migration (Pawan et al., 2012). As well, using the cloud for data storing needs to comply with several properties, such as security, reliability, and scalability, under limited Internet connection bandwidth (Mora et al., 2012).

To provide efficient access to distributed data and ensure a high degree of reliability, availability, and scalability Chang et al., (2008) proposed Bigtable system based on replicating not encrypted data without providing privacy and data security. An alternative mechanism is Hadoop and MapReduce based on splitting the data set into independent chunks processed in parallel and reducing them (Dean & Ghemawat, 2008). However, as shown in Herodotou et al., (2011), its main drawback is the low efficiency. In 2006, Google introduced the Google File System (GFS), a well-known solution to deal with reliability in Google's data centers. The basic idea is a redundant storage system (Ghemawat et al., 2003).

Not relational databases (NoSQL) consider the heterogeneity of unstructured data (Leavitt, 2010). However, the two most popular NoSQL databases, Cassandra, and MongoDB, have problems with data security and privacy (Okman et al., 2011).

Distributed Data Base (DDB) stores data on various computer network sites and uses logic to organize the set of data (Ozsu & Valduriez, 1991). There are two ways to construct DDBs. The top-down approach takes a database and distributes it over various sites, while the bottom-up approach unites distinct databases with one interface. The main field of application of DDBs is structured data storage. Therefore, it does not apply to arbitrary data sets, such as Big data.

Content Delivery Network (CDN) (Dilley et al., 2002) is a set of servers or Point of Presence (PoP) whose purpose is to provide faster content delivery. The following principles of CDN are important: load balancing, bandwidth conservation, and time efficiency. However, CDNs are not widely used in practice because they are not flexible with added cost and complexity.

1.1.2 Error correction codes

The error correction codes are based on Hamming's idea of adding additional data to detect and correct errors (Hamming, 1950). Depending on the application areas, approaches to building error correction systems differ. The balance between reliability and data redundancy is essential for storage systems since data redundancy affects the data volume and costs. The most expensive mechanism for ensuring reliability is data replication.

From another perspective, error correction codes and their modifications, such as erasure codes and regeneration codes, can provide greater reliability with lower redundancy than replication (Morelos-Zaragoza, 2006). An important issue when choosing an error correction code is the maximum number of errors that can be detected and corrected for given data redundancy (Krishna et al., 1992; J. Sun & Kirshna, 1992).

Watson & Hastings, (1966) present an alternative solution - modular error correction codes based on the Redundant Residue Number System (RRNS). The RRNS is a non-weighted system representing an integer as a set of residues obtained by dividing the original number by a set of coprime numbers. The sizes of the residues are smaller than the size of the original number. An additional advantage of RRNS for the design of distributed storage systems is that it is a secret sharing scheme that provides data security (Garner, 1959; Krishna et al., 1992).

However, there are two main problems: the computational and memory complexity of the data decoding algorithms. There are two widely used methods: Projection and Syndrome. The projection method is a universal method to detect and correct an error with any RRNS moduli. Its disadvantage is the exponential computational complexity depending on the number of correctable errors (Barsi & Maestrini, 1973; Watson & Hastings, 1966). The syndrome method reduces computational complexity to quadratic, but it requires storing large tables of constants in a memory (Fatt Tay & Chang, 2016; Yau & Liu, 1973).

To reduce the required memory, researchers can rely on two approaches. The first one uses auxiliary functions as an error syndrome, such as the rank of a number (N. Chervyakov et al., 2019). The second approach involves additional restrictions on RRNS modules. Other error correction codes provide the reliability of individual modules' storage.

The second problem is related to increasing the number of correctable errors with the same encoding parameters (Fatt Tay & Chang, 2016). There are several solutions to solve this problem. The first one is to unbalance RRNS moduli when one or more RRNS moduli are several times larger than the rest of the moduli (A. Tchernykh, Miranda-López, et al., 2019). This approach's disadvantage is related to the case when errors occur in the largest RRNS modulus so that the amount of incorrect data is significant and may exceed the threshold.

An alternative approach is a 2L-RRNS model. This model uses the same process for each level recursively, increasing the security of the system. The second level uses the small residue representation of the first level and creates smaller residues. The same or different moduli set can be used on both levels. Given that the second-level modules act as independent error correction codes, this approach allows correcting a larger number of errors than 1L-RRNS (P. Ali et al., 2014; Skavantzos & Abdallah, 1999), as well as reducing the computational complexity of decoding (Barati et al., 2008; Timarchi & Navi, 2007).

1.1.3 Distributed storage security

One of the main goals of Cloud technologies is to provide access to data at any time. Users get the opportunity to use Cloud services without involving specialists with simple and intuitive interfaces. Classical approaches to ensuring data integrity are based on identical or non-identical redundancy (storing copies or histories, respectively).

The task of ensuring data integrity is complex. It includes integrity control, maintenance, and recovery if data is violated for any reason. There are various ways to solve the problem of monitoring and ensuring data integrity. One of them is calculating the checksums and comparing them with the reference checksums (Menezes et al., 2018). Other methods are based on cryptographic techniques, key and keyless hashing, and electronic signature (Attas & Batrafi, 2011; Lillard et al., 2010; Wang & Yu, 2005). The disadvantage of these methods is the inability to ensure integrity without extra data for recovery mechanisms.

Redundancy is a widespread solution for ensuring data integrity. In Bhagwat et al., (2006), the authors claim that introducing redundancy into a storage system may improve data centers' reliability. They used different essential solutions, such as erasure correcting codes, failure detection, and recovery. However, hardware and software implementations of the Redundant Array of Independent Disks (RAID) (Chen et al., 1994; Z. Sun et al., 2018) have the disadvantage of the inability to control data and high redundancy.

Some methods control integrity by comparing the reference values and calculated hash codes (checksums) when requesting the data. However, the lack of mechanisms for data recovery does not allow ensuring integrity. Bowers et al., (2009) propose the HAIL system that uses so-called integrity-protected error-correcting codes (IP-ECC), achieving cross-server redundancy through ECC mechanisms. The authors also use proofs of retrievability (PoRS) to enable Cloud storage to prove that all stored data is available. Furthermore, if any failure is detected, the corrupted information is reallocated.

Abu-Libdeh et al., (2010) propose a Cloud storage proxy named RACS (Redundant Array of Cloud Storage). It uses RAID-like techniques. The main motivation is reducing the one-time cost of switching providers to prevent vendor lock-ins and better tolerate provider outages or failures.

Bessani et al., (2013) design a system called DEPSKY from a more commercial point of view. DEPSKY improves availability and confidentiality provided by Cloud storage. This system relies on Cloud-of-Clouds combining Byzantine quorum system protocols, cryptographic secret sharing, and erasure codes.

A recent approach named DROPS is presented by Ali et al., (2018). DROPS divides a file into several fragments, then replicates each fragment over the Cloud. The novelty of DROPS is to use a graph T-coloring technique that ensures that the pieces are stored at a certain distance to prevent an attacker from guessing the fragments' locations.

An alternative way is to use RRNS, which on the one hand, is an error correction code, which allows restoring the result when an error occurs, and on the other hand, is a secret sharing scheme that ensures data security (N. Chervyakov et al., 2019). In Celesti et al., (2016), the authors propose an approach that uses RRNS to code and divide data. Their experiments show the relation between encoded/original file sizes with upload/download access speeds. A. Tchernykh et al., (2018) show how the security of stored data depends on the RRNS parameters. On the other hand, a 2L-RRNS model can increase the number of detected and correctable errors compared to 1L-RRNS (Barati et al., 2008). Therefore, 2L-RRNS can ensure the reliability and integrity of stored data better than 1L-RRNS.

1.2 Problem statement

Cloud storage as a service model assumes a high trust level that may not be realistic in real-world applications environments. For example, several potential issues can arise if users depend on a single Cloud storage provider. Among them, we can mention reliability, safety, quality of service, financial costs, information leakage, and conspiracy. Besides, if the provider is unavailable or the contract established between the client and provider is canceled, the client will no longer access their data. A more critical situation appears if the Cloud providers suddenly go out of the market. In this case, users can permanently lose their data. In this context, one of the biggest challenges is to provide reliable and scalable data management.

Also, non-stationarity is one of the essential factors associated with the Cloud. The occurrence of technical failures, data security breaches, and collusions is challenging to predict and mitigate their consequences (N. Chervyakov et al., 2019; A. Tchernykh, Miranda-López, et al., 2019).

One possible direction is to rely on replication. This approach can ensure data availability, but failures may cause inconsistency among copies of the same information. From a business point of view, data leakage (Aora & Gupta, 2012; Silver-Greenberg et al., 2014; Weiss & Miller, 2015) is the primary concern for fully adopting Cloud services. It could lead to a negative impact on the trust of an organization.

Figure 1 shows significant inhibitors to Cloud adoption (Rao & Selvamini, 2015). It depicts that data leak prevention has 88% of the critical and essential inhibitors. Data segregation and protection are with a 92% impact on security inhibitors.

For example, Billing-as-a-Service is a business-to-business (B2B) cloud service that streamlines the billing process and offers a broad financial service scale (Carnahan, 2020). It allows users to safely manage their financial documents and perform different search and modification operations on the information. For the management of this type of service, appropriate data protection must be considered before data outsourcing to cloud providers.

Cryptography ensures data confidentiality, integrity, as well as authentication of only authorized users. However, its significant drawback is that it does not allow data processing. Moreover, high availability, one of the fundamental aspects of information security, cannot be ensured through the use of cryptography. Besides, maintaining the encryption keys secret and updated between participants is also challenging.

In the literature, works like DepSky (Bessani et al., 2013) and RACS (Abu-Libdeh et al., 2010) use distributed storage mechanisms based on secret sharing schemes and error correction codes to optimize the classical replication reducing the load of the transmission network.

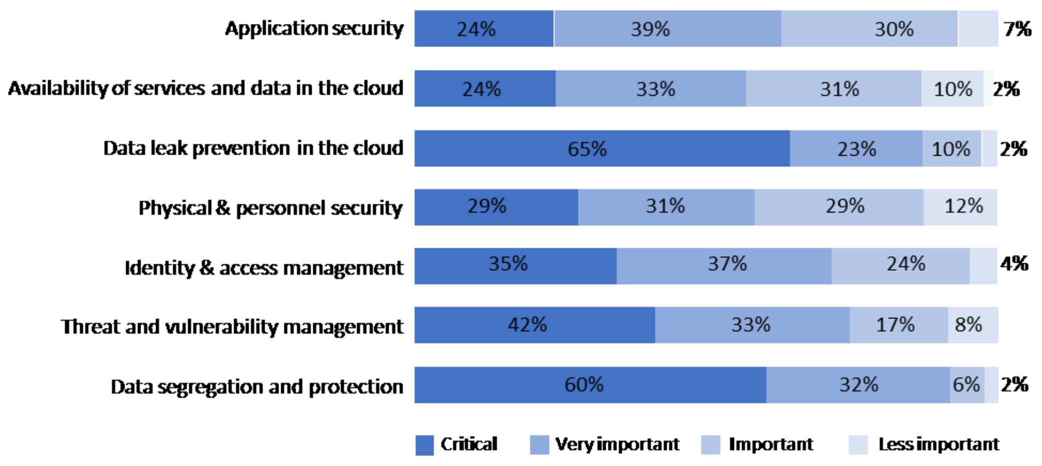


Figure 1. Data Security and Privacy - Major Inhibitor to Cloud Adoption (Rao & Selvamini, 2015).

RRNS is adapted to develop data storage systems (Celesti et al., 2016; N. Chervyakov et al., 2019). These systems perform the encoding process known as One-Level RRNS (1L-RRNS). Nevertheless, to continue improving the reliability, it is necessary to increase the number of moduli, which may increase the system's complexity and a security imbalance that may lead to information leakage.

Barati et al., (2008) proposed an alternative approach, a Two-Level RRNS architecture (2L-RRNS). The idea is to perform two encoding processes, one after the other. On the first step, 1L-RRNS is applied for initial data. On the second step, 1L-RRNS is applied for each residue using the same or different moduli set. Obtained small residues can increase calculations speed, decrease power consumption, increase security and fault tolerance. The main advantage is possibility to select reduced moduli set. This is because the dynamic range of the system, its speed, as well as its hardware complexity depend both on the form of the selected moduli and their number.

2L-RRNS provides favorable features for data integrity management in multi-cloud environments. Another importance characteristic is the mechanism to adapt its system behavior to the external conditions. Also it allows processing information without decoding and encoding unlike conventional encryption algorithms.Therefore, RRNS provides characteristics for designing configurable fault-tolerance storage in multi-cloud environments.

From the above broad, general-purpose statements, we narrow the focus to specific questions to be answered:

- Which mechanisms of RRNS-based Secret Sharing Schemes can improve reliability?
- Can the performance, reliability, and redundancy of the distributed storage be improved by adapting system parameters?
- What are the limitations of data storage in cloud environments?
- What cost comes from increased reliability?

1.3 Objective of the thesis

The general objective is to design a configurable mechanism based on a Redundant Residue Number System with the improved capability to detect and correct errors.

Several specific objectives deduced from a general objective are listed as follows:

- Design a novel 2Lbp-RRNS mechanism for error detection and correction as an extension of the classical 2L-RRNS based on Hamming distance and backpropagation.
- Provide analysis of the reliability upper bounds of the traditional threshold 2L-RRNS and new 2Lbp-RRNS solutions to estimate the number of detectable and correctable errors.
- Analyze 1L-RRNS, WA-RRNS, AR-RRNS, 2L-RRNS and 2Lbp-RRNS architectures considering redundancy, storing and extraction time, coding/decoding speeds, and access speeds in different scenarios.
- Evaluate and compare the behavior of state-of-the-art secret sharing schemes for data storage design such as Mignotte, Asmuth-Bloom, Reed-Solomon code+AES, etc.
- Evaluate the computational cost versus reliability.
- Design the experimental framework for a data storage system based on seven Cloud storages: DropBox, GoogleDrive, OneDrive, Sharefile, Box, Egnyte, and Salesforce.

1.4 Contribution

As a **scientific contribution**, we introduce novel two-level error detection and correction mechanism 2Lbp-RRNS (Section 3.7) based on the Residue Number System (Section 2.3.3) with improved the reliability of state of the art RRNS solutions (Section 2.3.3.5).

It combines elements of the Mignotte threshold secret sharing schemes (Section 2.5), 2L-RRNS model (Section 3.5.2), RRNS-based error correction codes (Section 2.4), and concepts of backpropagation and Hamming distance (Section 2.4.1).

In a two-level encoding scenario (Figure 15 and Figure 16, Section 3.7.1), the information is encoded by two consecutive steps, obtaining a set of (files) segments with the data represented by their corresponding residues.

The 2L-RRNS model of Barati et al., (2008) uses the Projection method as an error correction code. However, when the number of errors increased up to the number of control (redundant) RRNS moduli, the Projection method cannot detect errors and up to the half number of redundant moduli, correct errors even in a two-level architecture. To overcome this limitation, our solution uses the backpropagation and Hamming distance concepts.

For errors that 2L-RRNS cannot correct, 2Lbp-RRNS generates several potential solutions that CRT cannot verify. To validate which of these final solutions is correct, 2Lbp-RRNS uses the concept of backpropagation. . 2Lbp-RRNS encodes each potential solution back obtaining new encoded data. Then, it calculate the Hamming distance of each of these new encoded data and original erroneous data. The set with the smallest Hamming distance is the one that has enough correct segments (residues) to recover the information correcting the errors.

2Lbp-RRNS increments its detection and correction capability than 2L-RRNS or traditional 1L-RRNS models due to the capacity to determine which residues are correct from a potential solution. The scenario where 2Lbp-RRNS cannot perform correction occurs when two sets of possible solutions have the same minimum Hamming distance, so it cannot be determined which is the correct option.

1.5 Thesis outline

Chapter 2 introduces concepts, terminology, and definitions that we will use throughout the entire thesis. In this chapter, we describe the main mechanisms used in data security. We give the definition, parameters, and schemes based on the Residue Number Systems, like the Mignotte scheme, Modular Projection method for error detection-correction.

Chapter 3 provides the methodology framework used in this research work and a mathematical basis for the designed approaches. We describe a scheme based on a two-level RRNS to improve the system's reliability without compromising its security. Additionally, we explain the evaluation metrics used to assess the performance of the schemes.

Chapter 4 describes the experimental results obtained from evaluating the proposed schemes and their comparison with classical approaches. We drew several important conclusions from the results obtained in this chapter.

Finally, Chapter 5 provides general conclusions, contributions, and future work derived from this thesis.

Chapter 2. Background

To better understand the analysis and evaluation performed in this thesis, it is necessary to provide the background regarding Cloud computing, its security and reliability issues, secret sharing schemes, and encryption strategies presented in the literature.

2.1 Cloud Computing

Cloud computing is a well-known concept that supports an enormous pool of shared resources and services. The National Institute of Standards and Technology (NIST) defines the Cloud as: "a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management effort or service provider interaction "(US Department of Commerce 2013a).

The Cloud provides hosted services over the Internet. Users, as and when needed, on an hourly basis, can use any Cloud service. This "on-demand" approach makes the Cloud flexible. Three primary scenarios are used:

a) Infrastructure as a Service (IaaS). IaaS refers to on-demand provisioning of the hardware and software – servers, storage, networks, and operating systems.

b) Platform as a Service (PaaS). PaaS involves offering tools and services designed to make coding, test, and deploy custom applications. However, there are some limitations to the interoperability, programming support languages, or capabilities of using current platforms.

c) Software as a Service (SaaS). SaaS delivers special-purpose software that is remotely accessible by users through the Internet with a usage-based pricing model.

Nowadays, the Cloud storage service is gaining immense popularity. It is generalized as a service named Storage as a Service (STaaS) (Kulkarni et al., 2012). The Cloud owner manages this service supplying the user with access to data storage. Initially, companies saw STaaS as a cost-efficient way for small organizations that lacked personnel or storage infrastructure.

These services are deployed on four different models:

a) Private Cloud: This infrastructure is operated, and in some cases, managed by a private organization.

b) Community Cloud: Many organizations share the Cloud, and it is designed for a common concern among a specific community.

c) Public Cloud: This model is available to the public and is owned by an organization selling Cloud services, like Amazon Web Services.

Nevertheless, as Cloud services are implemented, the Cloud owners and clients face a series of risks and threats. Knowledge about these treats has to be the first step to prevent them. Hence, security is the main concern of several companies deciding how to use clouds.

2.1.1 Multi-Cloud approach

As the Cloud becomes popular, users are more reluctant to deal with a "single" Cloud storage due to potential issues such as service availability failure and malicious insiders. Hence, in recent years, users are moving towards a multi-cloud or cloud-of-cloud paradigm (Vukoli, 2010).

We can define multi-cloud as the services of different clouds combined into a single heterogeneous architecture. The services may or may not be connected or organized among themselves. We see it as the storage distribution across several Cloud architectures, see Figure 2.

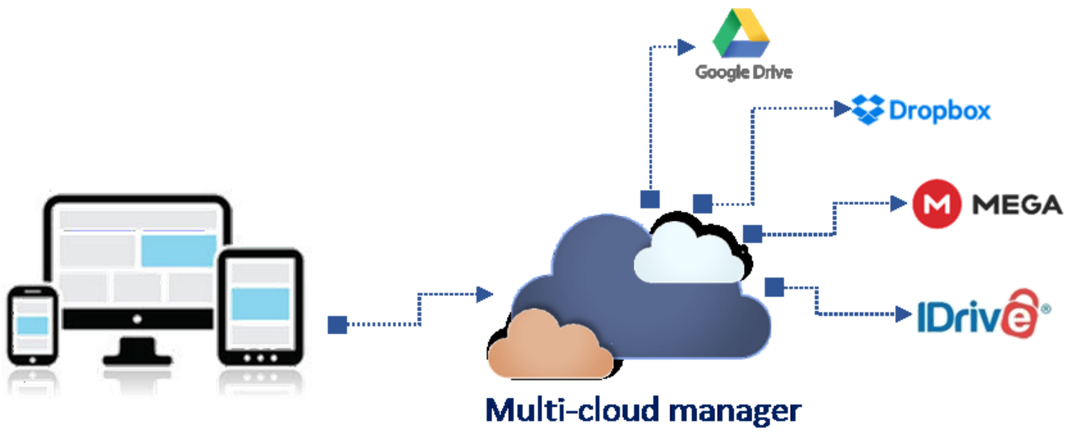


Figure 2. Multi-cloud architecture

There is a subtle difference with Hybrid Clouds, but there are important aspects that make multi-cloud the future of hybrid clouds:

1. Multi-cloud requires more attention in terms of security and management. The complexity of this model is much greater than the hybrid Cloud.

2. It involves many participants that need to be managed.

The integration process to multi-cloud architectures brings more complexity and adds significant value to an organization if the right providers are chosen to meet their specific requirements.

Today, there are many Cloud providers to choose from; see Table 1

Table 1. Major Cloud providers

Service Provider	**Names**
IaaS	Amazon EC2, Amazon S3, GoGrid
PaaS	Google App Engine, Microsoft Azure Services, Amazon Elastic Map Reduce
SaaS	Salesforce, Google Docs
sTaaS	Pure Storage, Bell Integrator, Zadara, NetApp, Dropbox, IDrive, Box

Currently, many Cloud storages with different characteristics are available. Each Cloud has its security mechanisms, reliability, and upload and download speeds. In Table 2, we listed some of the most common Cloud storages.

2.2 Security issues in Cloud storages

Security has been one of the most challenging problems for any IT, followed by concerns regarding compliance, privacy, and legal matters (Hashizume et al., 2013). In general, every Cloud storage system design must consider three crucial aspects: confidentiality, integrity, and availability. Several studies, including Sangroya et al., (2010), quote security as the primary level that confronts Cloud users.

Table 2. List of 24 Cloud storages

Cloud Storage	URL
Alibaba Cloud	https://www.alibabacloud.com/
Amazon Drive	https://www.amazon.com/gp/drive/about/
Box	http://box.com/
Certain Safe	https://certainsafe.com/
Dropbox	http://dropbox.com/
Egnyte	https://egnyte.com/
Elephant drive	https://home.elephantdrive.com/
Flip Drive	https://dlipdrive.com/
Google Drive	https://www.google.com/drive/
Hub spot	https://www.hubspot.com/
iCloud	https://www.icloud.com/
IDrive	https://www.idrive.com/
Jumpshare	https://jumpshare.com/
JungleDisk	https://www.jungledisk.com/
Justcloud	http://www.justcloud.com/
MediaFire	https://www.mediafire.com/
Mega	https://mega.nz/
pCloud	https://www.pcloud.com/
Rackspace	https://www.rackspace.com/cloud
Salesforce	https://www.salesforce.com/
Spideroak	https://spideroak.com/one/
SugarSync	https://www2.sugarsync.com/
Windows Azure	https://azure.microsoft.com/en-us/services/storage/
Yandex Disk	https://disk.yandex.com/

Classical security mechanisms like identity, authorization, and authentication are no longer enough for the cloud in their current form (Li & Ping, 2009). Risks of functionality and reliability also affect Cloud storage systems (A. Tchernykh et al., 2020). We underline some of the threads:

- Infrastructure threats: Denial of service, privacy leakage, misuse of resources, access control issues, account hijacking, etc.
- Data storage threads: Denial of service, data loss, disk errors, data modification, collusion, cyber-attacks, etc.
- Environmental threats: Earthquakes, floods, fires, etc.
- Deliberate threats: Interception, hackers attacks, etc.
- Accidental threats: PC errors, viruses, spam, etc.
- Unfairness: User errors, carelessness, falsification, curiosity, etc.

An error usually stands for faulty hardware, faulty software, technical malfunctions, DDoS attacks, human-made errors, providers' bankruptcy, etc.

On the other hand, a storage system can lost security and performance. According to the Cloud Security Alliance (CSA, 2010), several storage companies suffer from losing sensitive data in the last few years. For example, in November 2014, intruders took over personally identifiable information from Sony employees. In April 2011, due to technical failures, the majority of Amazon EC2 customers lost their data.

Therefore, it is essential to take special care to protect sensitive data. The immediate solution to this challenging situation is to use cryptographic methods, secret sharing schemes, erasure codes, and deduplication.

2.2.1 Security mechanisms

2.2.1.1 Digital signature

Digital signatures are based on asymmetric cryptography algorithms. They provide a layer of validation and security to information sent through a via that might not be secure, like, for instance, the Cloud. When properly implemented, a digital signature ensures that the right person sent his/her information. Digital signatures are equivalent to traditional handwritten signatures (R. Rivest et al., 1978). Some standard algorithms are RSA-based signature schemes, such as RSA-PSS (Bellare & Rogaway, 1998), Rabin signature algorithm (Rabin, 1979), and Pairing-based schemes such as BLS (Boneh et al., 2001).

2.2.1.2 Hash functions

Hash functions are a numeric value of a fixed length that uniquely identifies data. They represent large amounts of data by much smaller numeric values. It is widely used with digital signatures for verifying the integrity of the data sent through insecure channels. It is much easier to sign a hash value than to sign the original data. The mechanism is simple: the hash value of the received data is compared to the attached hash value to determine whether an intruder altered the data (Wang & Yu, 2005).

2.2.1.3 Encryption algorithms

Encryption algorithms are used to ensure sensitive data's privacy and confidentiality (Robling Denning, 1982). There are two types of key-based algorithms: symmetric and asymmetric. The Symmetric Key algorithms use the same key for encryption and decryption; hence, the participants must keep the secret key. The Asymmetric Key algorithms use two different keys, one for encryption and another key for decryption. Another difference between algorithms is the size of the segments (shares) on which the algorithm operates. Block algorithms are symmetric, meaning they work on larger blocks, usually 64 bits in length. On the other hand, Stream algorithms are symmetric and encrypt one bit (or byte) at a time.

Below, we list some of the most popular encryptions algorithms:

- Data Encryption Standard (DES) is an old (1972) symmetric block cipher that was designed by the NSA (National Security Agency) to serve as a standard for data encryption. The most exciting feature of DES is that the security is based on the key itself and not on the algorithm's secrecy. However, because the key size is small, brute force attacks have proven effective at cracking it. DES takes a fixed-length bit string of the original data and transforms it through a series of operations into an encrypted text bit of the same length. Each block size is 64 bits. There are 16 identical stages of processing, named rounds. There is an initial and final permutation.

- The RSA is a public-key algorithm proposed by Rivest et al., (1978). The encryption/decryption of RSA is very simple and effective. A secret message m is send, the receiver sends back his/her public key, which consists of two values, e and n. Later, the sender sends a ciphertext $c = m^e \bmod n$. The receiver decodes it using his/her private key d, computing $c = m^d \bmod n$ obtaining the original data. The security of RSA lies on generating a very large prime number difficult to factorize.

- The Advanced Encryption Standard (AES) is a symmetric-key algorithm based on the so-called Rijndael algorithm developed by Daemen & Rijmen, (2000). Specifically, AES was designed as a replacement for DES as its key size was too small. AES is a subset of Rijndael block cipher with three key lengths 128, 192, and 256 bits, and each block has a size of 128 bits.

  AES is based on a ‘substitution-permutation network.’ It performs all the computations on bytes; hence, it treats 128 bits of the plaintext block as 16 bytes. AES arranges the 16 bytes in a 4x4 matrix. The round depends on the key’s size; AES uses ten rounds for a 128-bit key up to 14 rounds for a 256-bit key. On each round, the algorithm calculates a new 128-bit round key from the original AES key.

The following Figure 3 depicts the general diagram of AES:

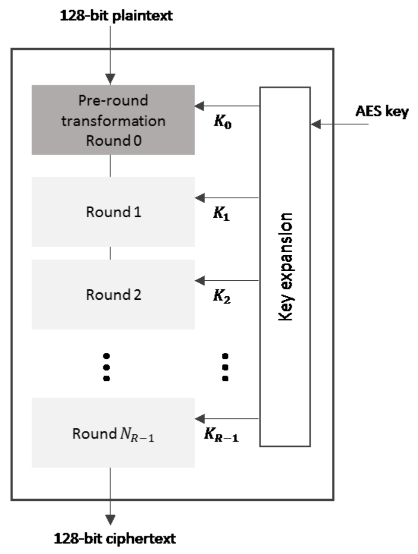


Figure 3. General diagram of the AES encryption algorithm

Each encryption round comprises four sub-processes, except the last round that does not require a new permutation. Figure 4 shows the four basic steps of each round. The first process (subBytes) substitutes each byte of the 4x4 matrix with another byte from the S-box lookup table. Next, the algorithm shifts each row to the left. It does not move the first column, it shifts the elements of the second column on one position, the third column on two positions, and finally, it shifts the last column elements on three positions. The next operation is MixColumns; the operation matrix transforms each byte of a column into a new byte. The result is another matrix consisting of 16 new bytes.

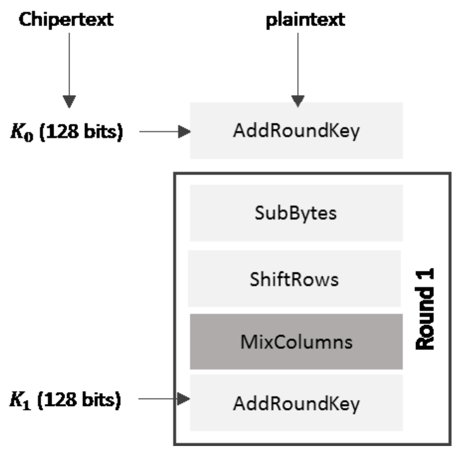


Figure 4. Encryption process for an AES round

Finally, the last step takes the 16 bytes of the matrix and performs an XOR operation with the round key's 128 bits. If this is the final round, the result is the ciphertext. Otherwise, it uses the resulting 128 bits as the input of the following round. However, efficient implementation and key management are needed to assure AES security since using a key size too long makes the algorithm complex to implement.

2.2.1.4 Homomorphic encryption

Rivest & Dertouzos, (1978) proposed the Homomorphic Encryption (HE), allowing carry out computations on ciphers generating an encrypted result which, when decrypted, matches the result of operations performed on the original numbers. They are based on an exponentiation operation and RSA function with additive and multiplicative homomorphic ciphers, respectively. Using these ideas of Gentry, (2009) proposes a wide range of fully homomorphic algorithms. HE allows computing a limited number of operations like additions and multiplications since encrypted data is mixed with noise increasing the security after each homomorphic operation. Therefore, there is a specific limit where this noise becomes too large to allow correct decryption.

2.3 Number Systems

In computer systems, numbers (representation of the data) are the basis of any operations. Since there is a trade-off between the word length and type of hardware, and between propagation delay and accuracy, several number representations have been proposed. The most popular are the Conventional Radix Number System and Signed-Digit Number System from the Fixed-Point Number System (Lu, 2004) and residue representations from the Residue Number System (Chang et al., 2015; Garner, 1959).

2.3.1 Properties of Number systems

In this section, we cover some of the basic properties of the Number systems (Szabo & Tanaka, 1967):

1. Range: The interval over which the system can represent every integer without having two numbers with the same representation.

2. Uniqueness: A number is unique if each number in the system has only one representation.

3. Redundancy: A number system is redundant if it uses more symbols than absolutely necessary to represent the number.

2.3.2 Weighted Number System

A Number system is weighted if there exists a set of weights w_i such that for any x it can be expressed in the form of $x = \sum_{i=1}^{n} a_i w_i$, where the a_i's are a set of permissible digits. If the values of w_i are successive powers of the same number, then the number system has a fixed base or fixed radix, e.g., the base 10 and base 2. If the weights are not powers of the same radix, then the systems are called mixed-radix systems (Szabo & Tanaka, 1967).

Weighted systems have many advantages, which we summarize as follows:

- Multiplication of division by a power of the base can be done easily by the shift operation.
- Magnitude comparison is relatively easy.
- Extending the number system range can be easily done by adding more digits.
- Overflow can be detected easily.

However, one of the weighted systems' most important limitations is the addition process's carry propagation problem. The Residue Number System solves this issue since the system can do additions 100% in parallel.

2.3.2.1 Fixed Radix Systems

Let us call a number fixed-radix or fixed-base system if it is a Weighted Number System (WNS) and the values of w_i are successive powers of the same number, e.g., the decimal number can be represented as

$X = (a_n \dots a_1) = \sum_{i=1}^{n} a_i w_i$ where $0 \leq a_i \leq 9$ and $w_i = 10^{i-1}$

Binary systems can be presented as

$X = (a_n \dots a_1) = \sum_{i=1}^{n} a_i w_i$ where $0 \leq a_i \leq 1$ and $w_i = 2^{i-1}$

2.3.2.2 Mixed Radix Systems

A conventional radix number N can be represented by n digits such as $(d_{n-1} d_{n-2} \dots d_1 d_0)_r$ where r is the radix, $d_i \leq i \leq n-1$ is a digit and each $d_i \in \{0,1,\dots,r-1\}$. This number system is a weighted positional system (Parhami, 1990). The number N can be represented by:

$$N = d_{n-1} \cdot w_{(n-1)} + d_{n-2} \cdot w_{(n-2)} + \cdots + d_0 \cdot w_0 = \sum_{i=0}^{n-1} d_i \cdot r^i \qquad (1)$$

with d_i being the weight of position i. If r is fixed, each $w_i = r^i$, but if r is not fixed, the number becomes a *mixed-radix number*.

2.3.3 Residue Number System

The Residue Number System (RNS) is based on a puzzle introduced by the Chinese mathematician Sun-Tzu, named as Chinese Remainder Theorem (CRT). Based on CRT, Garner, (1959) invented RNS in 1959. RNS is a non-weighted data representation system representing an integer number as a set of smaller numbers. Initially, the main application area of RNS was Digital Signal Processing (DSP). In modern cryptosystem design, the main RNS application is related to Montgomery modular multiplication, as it is well-suited to RNS arithmetic, since it avoids hard divisions.

Some of the main properties of RNS are the following:

- RNS handles carry-free and borrow-free operations.

- RNS supports fast, parallel arithmetic operations. In RNS, digit by digit computations can be performed independently since there is no ordering significance between the digits. Hence, RNS supports parallel computations. These advantages are very useful when the number of operations is increasing.

- RNS supports error detection and correction. An inherent property of RNS suggests that a Redundant RNS (RRNS) can be used for self-checking, error detection, and correction. Since there is no interaction between the residues, any error in a single module has a local effect, and errors can easily be detected and corrected. As modular arithmetic is performed in the residue representation, RRNS can correct arithmetical processing errors. This is a unique powerful capability that other correction codes do not have.

Even though RNS has some real drawbacks that without optimization, it can reduce a system's performance. These difficulties are associated with magnitude comparison, sign representation, overflow detection, data conversion, moduli selection.The division process is complicated due to the absence of a multiplicative inverse for the zero element and the fact that residue division and normal division correspond one to one only when the resulting quotient is an integer value. RNS can be defined in terms of a set positive integers $\{p_n, \dots, p_2, p_1\}$ called moduli set. Each p_i for $i = \overline{1, n}$ are called modulus. To avoid ambiguity on the recover data, the moduli of an RNS must be pairwise relative prime, where for each $\mathrm{GCD}(p_i, p_j) = 1$ for $i \neq j$ with GCD means greatest common divisor.

Each integer X, can be represented as a set of smaller integer called the residues. The residue set is denoted as $\{x_n, \dots, x_2, x_1\}$ where x_i is the $i - th$ residue. x_i is defined as the least positive remainder when X is divided by the modulus p_i. This relation can be written based on the congruence $x_i = X \bmod p_i$ also denoted as $x_i = |X|_{p_i}$. Such a representation is unique for any integer $X \in [0, P-1]$, where $P = p_n, \dots, p_2 p_1$ is known as the dynamic range. P is divided into two sub-ranges to represent a signed integer. The lower half and upper half ranges are used to represent positive and negative integers, respectively.

2.3.3.1 Forward conversion

Forward conversion is the process to transform from binary or decimal representation into residue notation. This process consists of the computations of residues by division operation, with the moduli as the divisors. The forward conversion complexity relies on the number of moduli employed and the magnitude of each modulus (Omondi & Premkumar, 2007). Suppose we want to convert a number X of n-bits with respect to a modulus p, $X = \sum_{j=0}^{n-1} x_j 2^j$, then:

$$|X|_p = \left|\sum_{j=0}^{n-1} x_j 2^j\right|_p = \left|\sum_{j=0}^{n-1} x_j \left|2^j\right|_p\right|_p \tag{2}$$

2.3.3.2 Reverse conversion

The reverse conversion is the process of transforming from residue representation back to conventional notation.

Chinese Remainder Theorem (CRT)

Given a moduli set $\{p_1, p_2, \dots, p_n\}$ consisting of pairwise relatively prime and an integer X represented by its residues $\{x_1, x_2, \dots, x_n\}$, where $x_i = |X|_{p_i}$, the number X and its residues are related by:

$$X = \left|\sum_{i=1}^{n} P_i \, x_i \left|P_i^{-1}\right|_{p_i}\right|_P \tag{3}$$

where $n > 1$, $P_i = {}^{P}/_{p_i}$, and $\left|P_i^{-1}\right|_{p_i}$ is the multiplicative inverse of $|P_i|_{p_i}$ defined by $\left|\left|P_i^{-1}\right|_{p_i} P_i\right|_{p_i} = 1$.

If the values involved are constrained so that the final value of X lies within the dynamic range P, then the modular reduction on the left-hand side can be ommited. We can rewrite X as:

$$\begin{aligned} X \cong \langle x_1, x_2, x_3, x_4, x_5 \rangle &= \langle x_1, 0,0,0,0 \rangle + \langle 0, x_2, 0,0,0 \rangle + \langle 0,0, x_3, 0,0 \rangle + \langle 0,0,0, x_4, 0 \rangle + \langle 0,0,0,0, x_5 \rangle \\ &\triangleq X_1 + X_2 + X_3 + X_4 + X_5 \end{aligned}$$

Now the reverse conversion consists of finding the values X_i's, which is much easy than obtaining X.

CRT requires a binary inner product operation followed by a large modulo P operation that is inefficient.

Example: Consider the moduli set $\{3,4,5\}$. We want to find the decimal representation of the residue set $\{2,3,1\}$ resulting from modulo reduction with respect to the given moduli. First, we determine $P_i's$, notice that P is the dynamic range which is equal to 60:

$P_1 = \frac{P}{p_1} = \frac{3 \cdot 4 \cdot 5}{3} = 20$, $P_2 = 15$, $P_3 = 12$

We calculate the multiplicative inverses of each P_i :

$P_1^{-1} = \left|P_1 \times P_1^{-1}\right|_3 = 1, \left|20 \times P_1^{-1}\right|_3 = 1,\ P_1^{-1} = 2,$

$P_2^{-1} = \left|P_2 \times P_2^{-1}\right|_4 = 1, \left|15 \times P_2^{-1}\right|_4 = 1,\ P_2^{-1} = 3,$

$P_3^{-1} = \left|P_3 \times P_3^{-1}\right|_5 = 1, \left|12 \times P_3^{-1}\right|_5 = 1,\ P_3^{-1} = 3,$

Finally, using equation (3):

$$X = \left|\sum_{i=1}^{3} P_i\, x_i \left|P_i^{-1}\right|_{p_i}\right|_P = |2 \cdot 20 \cdot 2 + 3 \cdot 15 \cdot 3 + 1 \cdot 12 \cdot 3|_{60} = 11$$

2.3.3.3 Mixed Radix Conversion (MRC)

Given a moduli set $\{p_1, p_2, \dots, p_n\}$ consisting of pairwise relative primes and an integer X represented by its residues $\{x_1, x_2, \dots, x_n\}$, where $x_i = |X|_{p_i}$, the number X and its residues can be uniquely represented in mixed-radix form as:

$$X = a_n p_{n-1} p_{n-2} \dots p_1 + \dots + a_3 p_2 p_1 + a_2 p_1 + a_1 \quad (4)$$

where $n > 1$, $0 \leq a_i \leq p_i$.

The Mixed-Radix Conversion establishes an association between the unweighted non-positional RNS and a weighted positional mixed-radix system. To recover the actual value of X is needed to obtain the Mixed-Radix digits (MR), a_i using equation (5):

$$a_1 = x_1 \tag{5}$$

$$a_2 = \left|(x_2 - a_1)\left|p_1^{-1}\right|_{p_2}\right|_{p_2}$$

$$a_3 = \left|\left((x_3 - a_1)\left|p_1^{-1}\right|_{p_3} - a_2\right)\left|p_2^{-1}\right|_{p_3}\right|_{p_3}$$

Meeting the constraint that the maximum weight contributed by the lower k digits must never exceed the positional weight of the $(k+1)$ digits, it can ensure unique representations.

Using the MR digits brings the next benefits: 1) the MR system is a weighted number system; therefore, magnitude comparison can be made easily, and 2) the MRC procedure requires operations moduli p_i only. Its major drawback is that the MR digits' computations have been strictly sequential and are not as parallel as the CRT (Szabo & Tanaka, 1967).

Example: Consider the moduli set $\{3,4,5\}$. We want to find the decimal representation of the residue set $\{2,3,1\}$ resulting from modulo reduction with respect to the given moduli. First, we determine the inverses $\left|p_1^{-1}\right|_{m_2}$ as follows:

$$\left|\left|p_1^{-1}\right|_{p_2} p_1\right|_{p_2} = 1, \left|\left|p_1^{-1}\right|_{p_2} \times 3\right|_4 = 1,\ \left|p_1^{-1}\right|_{p_2} = 3,$$

$$\left|\left|(p_2p_1)^{-1}\right|_{p_3}(p_2p_1)\right|_{p_3} = 1, \left|\left|(p_2p_1)^{-1}\right|_{p_3} \times 12\right|_5 = 1,\ \left|(p_2p_1)^{-1}\right|_{p_3} = 3,$$

Next, the mixed-radix digits are calculated as:

$$a_1 = x_1 = 2,$$

$$a_2 = \left| |p_1^{-1}|_{p_2} (x_2 - a_1) \right|_{p_2} = |3 \times (3 - 2)|_4 = 3,$$

$$a_3 = \left| |(p_2 p_1)^{-1}|_{p_3} (x_3 - (a_2 p_1 + a_1)) \right|_{p_3} = |3 \times (1 - (3 \times 3 + 2))|_5 = 0,$$

Therefore, the mixed-radix representation of the number is $\{2,3,0\}$. To obtain the real value, we apply equation (4):

$$X = a_3 p_2 p_1 + a_2 p_1 + a_1 = 0 \cdot 4 \cdot 3 + 3 \cdot 3 + 2 = 11$$

2.3.3.4 Moduli set selection

Choosing an appropriate scheme depends on the moduli set. The forward converters are classified based on the moduli set. Forward converters are based on arbitrary moduli set, or based on special or restricted moduli sets.

In the first type of converter, the moduli is consisted of prime numbers chosen in sequence until the selected dynamic range P is obtained. This moduli set allows us to achieve very large dynamic ranges without compromising the algorithm's performance.

The second type of converter is more efficient in terms of speed and power. This type of moduli is usually referred to as low-cost moduli-sets. Special moduli sets are based on power-of-two related moduli, for example, $\{2^n - 1, 2^n, 2^n + 1\}, \{2^n - 1, 2^n, 2^{n-1} - 1\}$. However, some applications may need a wide dynamic range that cannot efficiently achieve this moduli type.

The selection of the moduli set should follow some general rules:

1. The modulus should be coprime and small as possible, so that modulo reduction require minimum computational time.

2. To avoid overflow, the dynamic range should be large enough

2.3.3.5 RNS applications

RNS is widely known as an alternative non-positional number system that allows one to perform addition and multiplication operations fast and in parallel. It can accelerate speed of applications where these operations significantly decrease system performance. Well-known applications of RNS are Digital Signal Processing (DSP), cryptography, memory module, networking, and cloud computing.

In the area of Digital Signal Processing (DSP), finite impulse response (FIR) filters based on RNS were proposed by (Jenkins & Leon, 1977). In this type of filter, the analog input signal is converted to residues. The discrete convulsion operations are performed on the residues independently and later are converted to a binary representation. Jyothi et al., (2020) focused on increasing the FIR filter speeds for its application in artificial intelligence, machine language. In infinite impulse response (IIR) filters widely used in the control area (Bajard et al., 2011). The fault-tolerant FIR filters were proposed by Pontarelli et al., (2008). Their design not only responds to failures that occurred in the residues but also to possible errors in the process of the reverse converter avoiding the use of the trivial triple modular redundancy (TMR).

RNS is also used in network applications to solve some issues such as determining output port in a more straightforward way (Jia & Wang, 2013), increasing a wireless sensor network (WSN) lifetime (Campobello et al., 2012), enhancing their reliability by reducing the mean energy consumption of each sensor node, etc. Another RNS application was proposed by Junior et al., (2011), which utilizes the modularity of RNS to reduce the number of dropped messages in Ad Hoc networks caused by malicious nodes, node movement, and collision. Nazarov et al., (2018) designed RNS-based hardware for communication systems to protect from decoding errors during data transmission.

The issues of data confidentiality breach, loss of information, and unexpected termination of services by current cloud storage are also addressed by RNS (Celesti et al., 2016; N. Chervyakov et al., 2019; A. Tchernykh et al., 2018). In this case, it is used to split information on a finite number of residue segments and store them over different cloud providers to increase reliability.

2.4 Error detection and correction codes

2.4.1 Hamming weight and Hamming distance

Hamming code proposed by Hamming, (1950) is an error-correction code used to detect and correct errors that may occur when the data is moved from one point to another or stored. An important parameter of an error-detecting and correcting code is the *minimum Hamming distance*. Let suppose two binary n-tuples $v = (v_0, v_1, \dots, v_{n-1})$ and $w = (w_0, w_1, \dots, w_{n-1})$. On the one hand, the *Hamming distance* between v and w, denoted by $d(v, w)$, is the number of differences between the corresponding bits. On the other hand, the *Hamming weight* of v, denoted by $\mathscr{w}(v)$, is defined as the number of nonzero components of v.

The *minimum distance* denoted d_{min} is defined as the smallest Hamming distance between any pair of possible codevectors and can be calculated by: $d_{min} = min\{d(v, w): v, w \in C, v \neq w\}$, where *C* is a block code. So, a code's *minimum distance* defines how many errors bit we will be able to fix or correct. In any linear code, we have the notation (n, k, d), where d stands for the minimum distance achieved with the values of n and k (Lin & Costello, 1983).

2.4.2 RRNS error detection and correction

Defining an RRNS as an (n, k) code. All the integers in the legitimate range $[0, P_k)$ are legitimate (valid). All the n-tuple residue representations form an n-dimensional vector space, where the corresponding k-dimensional code space are valid. So, all residue representation in the n-dimensional code space (Ω) is a codevector that consists of an information part (k first residues) and a parity (redundancy) part (the remaining r residues).

An RRNS code has a minimum distance d, if and only if the product of the redundant modulus satisfies the following relation: $max\{\prod_{i=1}^{d-1} p_{j_i}\} \leq P_R < max\{\prod_{i=1}^{d} p_{j_i}\}$ where $1 \leq j_i \leq k + r$. $d \leq \mathrm{n} - \mathrm{k} + 1 = \mathrm{r} + 1$, which r is the redundant modulus. In Table 3, we can see the number of errors that can be detected and corrected for six different minimum distances.

Table 3. Number of detected and corrected errors regarding a minimum distance

Distance $\boldsymbol{d_{min}(v, w)}$	# errors detected	# errors corrected
1	0	0
2	1	0
3	2	1
4	3	1
5	4	2
6	5	2
$\boldsymbol{d}$	$(d-1) = r$	$\left\lfloor\frac{d-1}{2}\right\rfloor = \left\lfloor\frac{r}{2}\right\rfloor$

If in a codevector (residue representation), $X = \{x_1, x_2, \dots, x_n\}$ the i-th residue is faulty, it will result in the residue y_i, where $y_i \equiv x_i + e_i (mod\ p_i)$ $0 \le e_i < p_i$ and e_i being the error value. If $e_i = 0$, then there is not an error. Generally, the final codevector is $Y = X + E$, where E is the error vector and Y is the resulting one. The Hamming weight of E, $\mathcal{w}(E)$, also denoted by α, is the number of errors in Y.

To assess the error detecting and correcting capability of an RRNS system, the redundant moduli are equal to r, the system can detect r and correct $\left\lfloor\frac{r}{2}\right\rfloor$ errors (Chessa et al., 2004; Krishna et al., 1992; J. Sun & Kirshna, 1992).

Definition 1. The error capability, γ, of an RRNS code is the largest number of errors that may occur for which Y is not in Ω

Lemma 1. The value of γ is $d - 1 = r$.

Proof

For an RRNS with a minimum distance d, no error vector E having a weight $\mathcal{w}(E) = \alpha$, $0 < \alpha < d$, can change one codevector into another. For the triangular inequality if $d(Y, X) = \alpha$, $0 < \alpha < d$, then $d(Y, X_j) \ge d - \alpha > 0$ for all, $X_j \in \Omega$ and $X_j \ne X$. Therefore, Y cannot be a codevector. Also, there exist at least two codevectors, X_1 and X_2, such that $d = d(X_1, X_2)$. If $X_1 = X$, and $E = X_2 - X_1$, then $Y = X_2$. In such a case $Y \in \Omega$. Therefore, not all errors vectors having $\mathcal{w}(E) = d$ are detectable, thus the lemma is proved.

The decoding procedure is decode Y to a codevector $\hat{X}$ that differs from Y in the least number of places. A computationally impractical method is compare Y with every codevector in Ω . Then Y is decoded to that $\hat{X}$ that satisfies the condition $d\left(Y,\hat{X}\right) \leq d(Y,Z)$, for all $Z \in \Omega$, and $Z \neq \hat{X}$.

Definition 2. The error correcting capability, β, of an RRNS code is the largest number of errors that may occur for which correct decoding ($\hat{X} = X$) takes place.

Lemma 2. The value of β for an RRNS is $\beta = \left\lfloor\frac{d-1}{2}\right\rfloor = \left\lfloor\frac{r}{2}\right\rfloor$, where $\lfloor a \rfloor$ denotes the largest integer less tan or equal to a.

Proof

Let X_j be a codevector other than X in Ω. The Hamming distance among X, X_j, and Y satisfies the triangular inequality $d(Y,X) + d\left(Y,X_j\right) \geq d\left(X,X_j\right)$. Since $d\left(X,X_j\right) \geq d$ and $d(Y,X) = \alpha$, $d\left(Y,X_j\right) \geq (d-\alpha)$.

If $\alpha \leq \left\lfloor\frac{d-1}{2}\right\rfloor$, then $d - \alpha > \left\lfloor\frac{d-1}{2}\right\rfloor \geq \alpha$. Thus $d\left(Y,X_j\right) \geq d(Y,X)$ therefore implying that Y is closer to X than any other codevector in Ω. On the other hand, when $\alpha > \beta$, we can show that there exists at least one codevector X_j such that $d\left(Y,X_j\right) < d(Y,X)$. In this case, incorrect decoding will take place. The lemma is proved.

2.4.2.1 Modular projection method

The projection method is introduced by Szabo & Tanaka, (1967). For detecting r errors, it needs r redundant moduli. The number of projections, in the worst-case scenario, is C_n^{k+1}.

Let $X \xrightarrow{RRNS} (x_1, x_2, \dots, x_k, \dots, x_{k+r})$ be the residue representation of an integer X with moduli set $\{p_1, p_2, \dots, p_k, \dots, p_{k+r}\}$ satisfying the conditions that without loss of generality, the moduli are sorted in ascending order, and $\forall i \neq j{:}\,\gcd\left(p_i, p_j\right) = 1$. There are two cases to consider: 1) X has no error if $X \in [0,P)$, where $P = \prod_{i=1}^{k} p_i$; 2) if t moduli are excluded, where $t \leq r$ the value of X will not be changed if represented in RRNS. Specifically, the criterion for error detection is if the recovered value of X falls out of the dynamic range P.

The main idea of this method is that it leaves out a residue from the given RNS-representation. Then, $X_i = |X|_{\bar{P}/p_i}$ where $\bar{P} = \prod_{i=1}^{n} p_i$ is said to be the $p_i - projection$ of X. X_i can be interpreted as the residue representation of X in a reduced $(k, n-1)$-RRNS with the $i-th$ residue x_i deleted.

Then, a $X_\wedge - projection$ of X, defined by $X_\wedge = |X|_{\bar{P}/p_\wedge}$, where $p_\wedge = \prod_{m=1}^{\lambda} p_{i_m}$, $\wedge = \{i_1, .., i_\lambda\}$, and $\lambda \leq (n-k) = r$. Hence, $X_\wedge$ can also be represented as a reduced residue representation of X, since residues $x_1, \ldots, x_\lambda$ are deleted. So, X can be represented by a $(k, n-k)$-RRNS. The new legitimate range is $[0, P')$ and the illegitimate range is $[P', \bar{P}/p_\wedge)$. Therefore, the $p_\wedge - projection$ of any legitimate X is still legitimate, provided that a sufficiently high dynamic range is retained in the $(k, n-k)$-RRNS to unambiguously represent X.

Suppose that X has an error, represented by $\tilde{X}$. Also, suppose that $\tilde{X}_i$ is a correct projection of $\tilde{X}$, then all other projections, $\tilde{X}_j$ $i \neq j$ are faulty. Therefore, the correct value of X is $|\tilde{X}_i|_{p_i}$.

Example. Consider the next parameters, $k = 3$ and $r = 2$ a moduli set $\{2,3,5,7,11\}$. The dynamic range is $P = 2 \cdot 3 \cdot 5 = 30$. The total RRNS range is $\bar{P} = P \cdot 7 \cdot 11 = 2310$. Suppose that an error occurs in the fourth residue resulting in $\bar{X} \xrightarrow{RRNS} (1,2,2,\mathbf{5},6)$ instead of $X = 17 \xrightarrow{RRNS} (1,2,2,3,6)$.

To correct the error, suppose we calculate each $p_i - projection$ removing the first moduli, then the second, and so on:

$\overline{X_1} \xrightarrow{RRNS} (2,2,\mathbf{5},6)$ based on the moduli set $\{3,5,7,11\}$, then $\bar{P}_{I_1} = \prod_{i=1}^{4} p_i = 1155$.

$\overline{X_1} = |2 \cdot 385 + 2 \cdot 231 + 5 \cdot 330 + 6 \cdot 210|_{1155} = 677 > 30.$

$\overline{X_2} \xrightarrow{RRNS} (1,2,\mathbf{5},6)$ on the moduli set $\{2,5,7,11\}$, then $\bar{P}_{I_1} = \prod_{i=1}^{4} p_i = 770$.

$\overline{X_2} = |1 \cdot 385 + 2 \cdot 616 + 5 \cdot 330 + 6 \cdot 210|_{770} = 677 > 30.$

$\overline{X_3} \xrightarrow{RRNS} (1,2,\mathbf{5},6)$ on the moduli set $\{2,3,7,11\}$, then $\bar{P}_{I_1} = \prod_{i=1}^{4} p_i = 462$.

$\overline{X_3} = |1 \cdot 231 + 2 \cdot 154 + 5 \cdot 330 + 6 \cdot 210|_{462} = 215 > 30$.

$\overline{X_4} \xrightarrow{RRNS} (1,2,2,6)$ on the moduli set $\{2,3,5,11\}$, then $\bar{P}_{I_1} = \prod_{i=1}^{4} p_i = 462$.

$\overline{X_4} = |1 \cdot 165 + 2 \cdot 220 + 2 \cdot 66 + 6 \cdot 210|_{462} = 17 < 30$. Since, $\overline{X_4} \in [0, P)$, meaning that the error was on the fourth residue on modulus 7, then the correct value is $x_4 = |17|_7 = 3$.

2.4.2.2 Syndrome method

The syndrome method is based on the Base Extension algorithm of Yau & Liu, (1973). To check the correctness of the result, it is necessary to compute the values $x'_{k+1}, x'_{k+2}, \dots, x'_{k+r}$ from $x_1, x_2, .., x_k$ by the base extension algorithm and compare them to the initial values $x_1, x_2, \dots, x_{k+r}$.

To detect an error, the following procedure is used. Let $X \xrightarrow{RRNS} (x_1, x_2, \dots, x_k, \dots, x_{k+r})$, with the moduli set $\{p_1, p_2, \dots, p_k, \dots, p_{k+r}\}$ and $\forall i \neq j \colon \gcd(p_i, p_j) = 1$. Then there are two cases, 1) if an element of x_i occurs, where $j \geq k+1$ then the number of nonzero Δ_c, $c = k+1, k+2, \dots, k+r$, are not greater than $\lfloor r/2 \rfloor$ and X^* is correct, where Δ_c- is a syndrome error. 2). If an element of x_i occurs, where $j \leq k$, then the number of nonzero Δ_c for all $c = k+1, k+2, \dots, k+r$ where $\Delta_c = |X^* - x_c|_{p_c}$, $X^* = \left|\sum_{i=1}^{k} P_i b_i x_i\right|_p$ and $\forall i = \overline{1,k}$ $P_i = P/p_i$ where b_i is the multiplicative inverse of $|P_i|_{p_i}$.

Using the approach of Fatt Tay & Chang, (2016), which splits the set of possible errors into three parts:

$$E_A = \left(e_{a_1}, e_{a_2}, \dots, e_{a_i}, 0, \dots, 0\right),$$
$$E_B = \left(0, \dots, 0, e_{b_1}, e_{b_2}, \dots, e_{b_j}, 0, \dots, 0\right)$$
$$E_C = \left(0, \dots, 0, e_{c_1}, e_{c_2}, \dots, e_{c_l}\right)$$

where $i, j, l < \left\lfloor \frac{r}{2} \right\rfloor$. We compute syndromes $\delta_1, \delta_2, \delta_3$, $\delta_1 = \left||X|_{P_A} - X\right|_{P_A}$, $\delta_2 = \left||X|_{P_B} - X\right|_{P_B}$, $\delta_3 = \left||X|_{P_C} - X\right|_{P_C}$, where $P_A = \prod_{p=1}^{i} p_{a_m}$, $P_B = \prod_{p=1}^{j} p_{b_m}$ and $P_C = \prod_{p=1}^{l} p_{c_m}$.

If $\delta_1 = \delta_2 = \delta_3 = 0$, there are no errors, and the result is correct. If any arbitrary syndrome is not equal to zero; and, the number of errors is greater than r, it is impossible to recover the right result.

2.4.3 Replication technique

Data replication is generally used to manage a great deal of data by creating identical copies of it in geographically distributed sites called replicas (Lamehamedi & Szymanski, 2007). The advantage of data replication is speeding up data access, reducing access latency, and increasing data availability. It is crucial to guarantee the availability of the replicas and data integrity features for a distributed system. There are two types of data replication: static and dynamic.

- *Static replication.* It follows deterministic policies. Therefore, the number of replicas and the host node is well defined and predetermined. Besides, these strategies are easy to implement, but they do not fully adapt to the Cloud environment.

- *Dynamic replication.* This kind of replication automatically creates and deletes replicas according to changes in the user access pattern, storage capacity, and bandwidth (Wei et al., 2010).

Nevertheless, the replication technique has some drawbacks, such as difficulty collecting run time information of all the data nodes in complex Cloud infrastructure and maintaining the data file consistency. The replica creation method finds the best time to create a new replica. An access recorder is then assigned to each data node, which is used to store the number of simultaneous users accesses to each file, including file name, several concurrent access, file size.

2.4.4 Erasure codes

The chance of a failure increases along with systems grows. Single parity used in RAID systems no longer are sufficient, and k-replication is too much waste of storage space. Erasure codes have been used in storage systems as an alternative to replication (Aguilera et al., 2005). Proper use of EC provides greater space efficiency and more level of protection but with higher complexity. Essentially, an EC is an error correction mechanism for bit erasures rather than bit errors. It transforms a message of k symbols into a longer message with n symbols such that from a subset of the n symbols we can recover the original data (Dimakis et al., 2010). EC has a code rate $= {}^{k}/_{n}$. Since EC is not homomorphic, it is suitable for building reliable distributed data storage system but does not allow efficient data processing. For EC systems, a common practice is to generate another encoded block instead of repairing a faulty node to reconstruct the completely encoded data.

2.4.5 Reed-Solomon codes

Reed-Solomon (RS) is a cyclic error-correcting code invented by Reed & Solomon, (1960). RS codes' idea is to describe a systematic way of building codes that could detect and correct multiple symbol errors. In a system, there are n disks or clouds, k of them hold the original data, and the remaining r hold coding (parity) information, which is calculated from the data. Since RS is a type of erasure code, if the erasures are known beforehand, RS can correct up to r deletions.

Basically, RS codes are MDS codes (Maximum Distance Separable), which have the property that if any r disks or clouds fail, RS can reconstruct the original data. On the one hand, it does not matter how many bits in a symbol are incorrect; RS only counts them as a single error. Reed-Solomon code is usually a poor choice if the data have random single-bit errors (MacWilliams & Sloane, 1977; Singh, 2013).

The RS's quadratic decoding time becomes unacceptable when it is used for vast data distribution over the internet due to data rates are of MB/s order (Forney, 2003). Reed-Solomon is an effective way to protect data integrity, and it is an excellent solution wherever small data blocks need to be verified, such as NASA standard (255;233;33) code. Moreover, RS codes are block codes; then, their implementation requires a previous evaluation of the transmission channel's erasure probability.

2.4.5.1 Basic Reed-Solomon

The encoder takes k data symbols of size s-bit word, where s must be large enough that $n \leq 2^s + 1$. s is typically constrained so that words fall on machine word boundaries: $s \in \{8,16,32,64\}$. Most implementations choose $s = 8$, since their systems contain fewer than 256 disks or clouds, and $s = 8$ performs the best. RS treats each word as a number between 0 and $2^s - 1$, and operates under *Galois*

Field arithmetic ($GF(2^s)$). Where Galois Field is a field with finite number of elements. The number of elements of any finite is a power p^n of a prime number p, which is the characteristic of this field. It is denoted by $GF(p^n)$ or by F_{p^n}. $GF(p^n)$ As with any field, a $GF(p^n)$ is a set on which the operations of multiplication, addition, subtraction and division are defined and satisfy the properties of commutative, distributive, and associative and there exists the neutral element of addition and multiplication (Mullen & Panario, 2013).

In the encoding phase, a Generator matrix or a Distribution matrix is constructed from a Vandermonde matrix, see Figure 5.

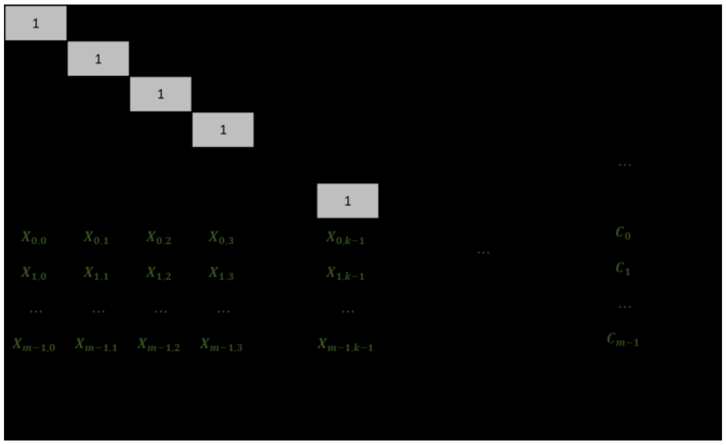

1
1
1
1
…
1
$X_{0,0}$ $X_{0,1}$ $X_{0,2}$ $X_{0,3}$ $X_{0,k-1}$
$X_{1,0}$ $X_{1,1}$ $X_{1,2}$ $X_{1,3}$ $X_{1,k-1}$
…
$X_{m-1,0}$ $X_{m-1,1}$ $X_{m-1,2}$ $X_{m-1,3}$ $X_{m-1,k-1}$
C_0
C_1
…
C_{m-1}

Figure 5. Matrix vector product to describe a coding system

The top k rows of the distribution matrix compose a $k \times k$ identity matrix. The rest of the rows (m rows) are called the coding matrix, creating a codeword, see Figure 6. The distribution matrix is multiplied by a vector that contains the data words and yields a product vector containing both the data and the coding symbols. Hence, to encode, RS needs to perform m dot products of the distribution matrix with the data.

When an erasure or error occurs, Reed-Solomon detects and corrects them in the following way. First, RS calculates a serial syndrome to check if the codeword is valid or not. If no error has occurred, the decoding process ends; otherwise, RS calculates the error-location polynomial and error-evaluator polynomial. Finally, when the values and locations of the errors are obtained using, for example, the Chien search and Forney algorithm, the codeword can be corrected by an XOR operation (Lin & Costello, 1983).

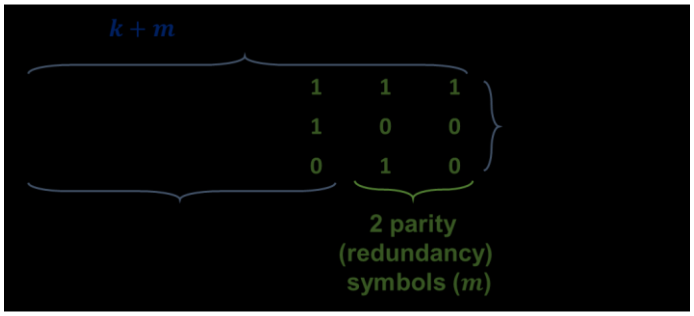


Figure 6. Example of a basic structure of Reed-Solomon codeword

In $GF(2^s)$, addition is equivalent to bitwise exclusive-or (XOR), and multiplication is more complex, typically implemented with multiplication tables or discrete logarithm tables. Naturally, we can expect a quadratic decoding time for Reed-Solomon's decoding process (Forney, 2003). For this reason, Reed-Solomon codes are considered expensive.

2.4.5.2 Cauchy Reed-Solomon (CRS)

The Cauchy Reed-Solomon modifies the classic RS in two ways. The first modification creates the distribution matrix differently using Cauchy matrices instead of Vandermonde matrices. The second one eliminates expensive multiplications operations by converting them to other XOR operations. This latter modification can be applied to Vandermonde-based RS codes. This last modification transforms G^T from a $n * k$ matrix of s-bit words to a $wn * wk$ matrix of bits. Again, s must be selected so that $n \leq 2^s + 1$ (Plank et al., 2009).

It is assumed that the disks are composed of w packets of equal size to implement these codes. Now each packet is calculated to be the bitwise XOR some subset of the other packets. This process is illustrated in Figure 7.

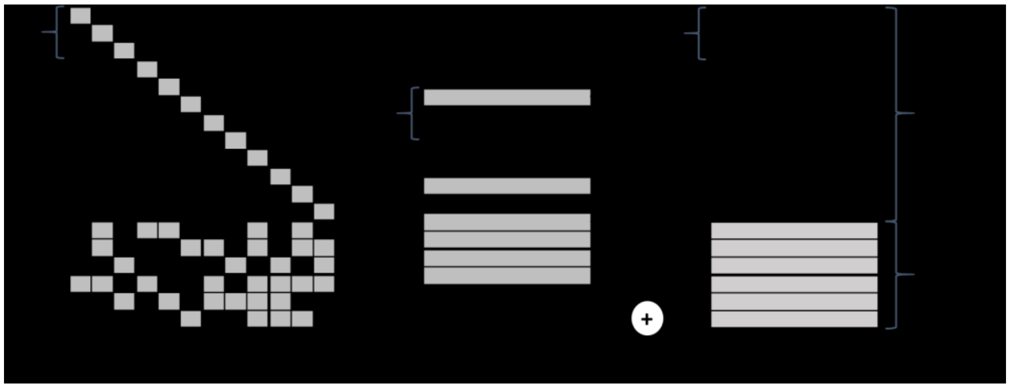


Figure 7. Cauchy Reed-Solomon coding process

2.5 Secret Sharing Schemes

A Secret Sharing Scheme (SSS) is a method where a dealer has a *secret,* and from it derives individual *shares* (shadows) that are distributed to different participants. Only an authorized subset of participants may recover the secret. The *access structure* $\mathcal{A}$-SSS of an SSS is a method of generating $\left(S, (S_1, \dots, S_n)\right)$ where S is the secret and $S_1, \dots, S_n$ will be the shares. Its elements are referred to as an authorized set and the unauthorized rest sets. Ito et al., (1989) remark that if a group can recover the secret, so can a larger group.

Depending on the quantity of information is leaked to an unauthorized set, SSS can be classified as:

- *Perfect secret*. In this scheme, any unauthorized set shares cannot give any information about the original data.
- *Computational-secure*. In this scheme, some information can be leaked to an unauthorized set. However, finding the secret is intractable, meaning there is no polynomial deterministic/randomized algorithm for solving it.

Researchers initially focused on these techniques on problems of secure information storage. However, SSS has found numerous other applications in cryptography and distributed computing. Nonetheless, the shares' size is exponential in the number of participants, which is a big drawback of this technique. The efficiency and security decrease as the quantity of the information that must be kept secret increases. Therefore, it is important for an SSS the size of the shares.

The entropy is used for analyzing the amount of information in a secret sharing. The entropy $H(\mathcal{X})$ can be interpreted as the average number of bits required for representing an element of $\mathcal{X}$ concerning some probability distribution (Gray, 2011). Hence, $H(S)$ can be viewed as the size of the secret and $H(S_i)$ as the size of the i −share. For a perfect threshold SSS, the relation $H(S) \leq H(S_i)$ holds true for all $1 \leq i \leq n$ (Karnin et al., 1983).

In the first threshold secret sharing scheme, only the number of participants in the recovery phase was essential for recovering the information. Let $n \geq 2, 2 \leq k \leq n$. The access structure $\mathcal{A} = \{A \in \mathcal{P}(\{1,2, \dots, n\}) |\, |A| \geq k\}$ is referred to as (k, n)-threshold access structure.

The formal definition of a threshold-SSS introduced independently by Shamir, (1979) and Blakley, (1979) is a (k, n)-secret sharing scheme is used to distribute a secret d among n participants such that any coalition of size k or more can construct d but smaller coalitions cannot.

The scheme of Shamir, (1979) is based on polynomial interpolation. Given any k pairs $(x_1, y_1), \dots, (x_k, y_k)$ with $x_i \neq x_j$ for all $1 \leq i < j \leq k$, there is one and only one polynomial $P(x)$ of degree $k - 1$ such that $P(x_i) = y_i$ for all $1 \leq i \leq k$. To obtain a (k, n)-SSS, a random polynomial $P(x) = a_{k-1}x^{k-1} + a_{k-2}x^{k-2} + \cdots + a_0$ is generated over $\mathbb{Z}_p[x]$, where p is a prime number $a_0 = d$ is the secret. The secret can be recover using Lagrange interpolation if k or more participants join by $S = \sum_{i \in A} \left(I_i \cdot \prod_{j \in A \setminus \{i\}} \frac{x_j}{x_j - x_i} \right)$. This scheme is ideal, since the size of the share does not exceed the size of the secret.

The secret sharing proposed by Blakley, (1979) uses hyperplanes theory to encrypt/decrypt the secret. The shares are any n distinct $(k - 1)$-dimensional hyperplanes that contain the secret, where and $(k - 1)$-dimensional hyperplane is a set of form $\left\{(x_1, \dots, x_k) \in \mathrm{GF}_q^k | \alpha_1 \cdot x_1 + \cdots + \alpha_k \cdot x_k = \beta\right\}$ where $\alpha_1, \dots, \alpha_k$, β are arbitrary elements of the field GF_q. The secret can be obtained by intersecting any k shares.

Two other secret sharing schemes fundamentally different are Asmuth-Bloom and Mignotte (Asmuth & Bloom, 1983; Mignotte, 1983). Both use CRT to recover the original secret.

.5.1 Mignotte Secret Sharing Scheme

Mignotte (k, n)-threshold SSS uses a sequence of pairwise coprime integers $p_1, p_2, \dots, p_n$ as parameters satisfying the equation (6):

$$\prod_{i=1}^{k} p_i > \prod_{i=1}^{k-1} p_{n-k+1+i} \tag{6}$$

That is, the product of the smallest k numbers is larger than the product of the largest $k-1$ numbers. Where $= \prod_{i=1}^{k} p_i$, $P_0 = \prod_{i=1}^{k-1} p_{n-k+1+i}$. The secret s is selected so that $P > s > P_o$. Each participant p_i holds a share s_i, where $s_i = |S|_{p_i}$. Since each element p_i are coprime, we can get a unique value S, where $0 \leq S < \prod_{i=1}^{k} p_i$. One big drawback of the Mignotte scheme is that it is not *perfect*, but can obtain small shares and can be used in applications in which the compactness of the shares is a major factor.

2.5.2 Asmuth-Bloom Secret Sharing Scheme

The Asmuth-Bloom (k, n)-threshold SSS uses special increasing sequences of pairwise coprime integers $p_0, p_1, p_2, \dots, p_n$ satisfying the equation (7):

$$\prod_{i=1}^{k} p_i > p_0 \cdot \prod_{i=1}^{k-1} p_{n-k+1+i} \tag{7}$$

In this scheme, the original secret $s_o$is selected from $\mathbb{Z}_p$. The shared secret s_0' is calculated as $s_0' = s_0 + \alpha p$, where α is a randomly chosen integer such that $0 \leq s_0' < P$. To recover the secret s_0 a coalition of at least k shares can be used to calculating $s_0 = |s_0'|_p$, where s_0' is calculated by the CRT.

2.5.3 AR-RRNS Secret Sharing Scheme

AR-RRNS is a residue-to-binary conversion method based on N. Chervyakov et al., (2017), where the authors propose an improved approximate CRT. Considering a number X in RRNS, if it is divided by the dynamic range P, the following variant of the CRT can be obtained:

$$\tilde{X} = \frac{X}{P} = \left|\sum_{i=1}^{n} \frac{\left|P_i^{-1}\right|_{p_i}}{p_i} x_i\right|_1 = \left|\sum_{i=1}^{n} k_i x_i\right|_1 \tag{8}$$

where $k_i = \frac{\left|P_i^{-1}\right|_{p_i}}{p_i}$, $i = 1,2,..n$. Then, the value of each sum of (8) is in the range of $[0,1)$, which provides sufficient information to assess the sign and magnitude of the RRNS number. Therefore, the approximate CRT substitutes the number by its fraction, adjusting the representation accuracy based on available resources. The value $\tilde{X}$ can be considered as a positional characteristic of X, where:

$$X = \tilde{X}P \tag{9}$$

Based on this approximate CRT, the approach may reduce the number of calculated projections of the number and replaces the computationally complex operation of long integers' division by taking the least significant bits. Based on equations (8) and (9), the rank of the number is computed by equation (10) :

$$X = \sum_{i=1}^{n} P_i \left|P_i^{-1}\right|_{p_i} x_i - R_x \cdot P \tag{10}$$

where $r_x = \left\lfloor \sum_{i=1}^{n} \frac{\left|P_i^{-1}\right|_{p_i}}{p_i} x_i \right\rfloor$ and $P = \prod_{i=1}^{n} p_i$, $P_i = \frac{P}{p_i}$. For all $i = \overline{1,n}$, and r_x is a positive integer representing the rank of X, showing how many times the dynamic range of the RRNS can be increased.

The integer X can be recover unequivocally if $N = \lceil \log_2 \rho \rceil$, then $r_x = R_x$ or $r_x = R_x - 1$, where $R_x = \left\lfloor \sum_{i=1}^{n} \frac{k_i x_i}{2^N} \right\rfloor$, $k_i = \left\lceil \frac{\left|P_i^{-1}\right|_{p_i} 2^N}{p_i} \right\rceil$, and $\rho = \sum_{i=1}^{n} p_i - n$.

Example: Let RRNS moduli set be $\{2,3,5\}$, the dynamic range is $P = 2 \cdot 3 \cdot 5 = 30$. Suppose two intenger $X = 8 \xrightarrow{RRNS} (0,2,3)$, and $Y = 29 \xrightarrow{RRNS} (1,2,4)$. Next, it is calculated the required parameters:

$P_i = \frac{P}{p_i},\ P_1 = 15,\ P_2 = 10,\ P_3 = 6;\qquad \beta_i = \left|P_i^{-1}\right|_i P_i,\ \beta_1 = 15,\ \beta_2 = 10,\ \beta_3 = 6;$

$\rho = -3 + 2 + 3 + 5 = 7,\ N = \lceil \log_2 7 \rceil = 3;\quad k_i = \left\lceil \left|P_i^{-1}\right|_i \frac{2^N}{p_i} \right\rceil,\ k_1 = 4,\ k_2 = 3,\ k_3 = 2;$

The values of X and Y are calculated:

$\sum_{i=1}^{3} k_i x_i = 4 \cdot 0 + 3 \cdot 2 + 2 \cdot 3 = 12,\qquad \sum_{i=1}^{3} k_i y_i = 4 \cdot 1 + 3 \cdot 2 + 2 \cdot 4 = 18;$

$R_X = \left\lfloor \frac{\sum_{i=1}^{3} k_i x_i}{2^N} \right\rfloor = 1,\ R_Y = \left\lfloor \frac{\sum_{i=1}^{3} k_i y_i}{2^N} \right\rfloor = 2\ ;$

Finally, $X = \sum_{i=1}^{3} P_i \left|P_i^{-1}\right|_{p_i} x_i - R_X \cdot P = (15 \cdot 0 + 10 \cdot 2 + 6 \cdot 3) - (1 \cdot 30) = 8$

$Y^* = \sum_{i=1}^{3} P_i \left|P_i^{-1}\right|_{p_i} y_i - R_Y \cdot P = (15 \cdot 1 + 10 \cdot 2 + 6 \cdot 4) - (2 \cdot 30) = -1$

Since $Y^* < 0$, then $Y = Y^* + P = -1 + 30 = 29$

Therefore, if the approximate rank R_X of X equals to $r_X + 1$, then the value is less than zero. In this case, the dynamic range is added to the negative value and obtains the real X.

Since AR-RRNS is based on RRNS, it has error detection and correction capability. The error correction method based on the approximate CRT is named AR-ECC. This method is derived from equation (10) where $\sum_{i=1}^{n} k_i x_i$ can be represented by the equation:

$$\sum_{i=1}^{n} k_i x_i = X + r_x P \tag{11}$$

The idea of error-correcting is the next, assuming that an error $E \xrightarrow{RRNS} (e_1, e_2, \dots, e_n)$ occurred resulting in a value $X^* = X + E$ instead of X. Then by equation (12):

$$\sum_{i=1}^{n} k_i(x_i + e_i) = X + E + r_x P + r_E P \qquad (12)$$

Since $X < \prod_{i=1}^{k} p_i = R$. The value of $\lfloor E/R \rfloor$ is $\left\lfloor \frac{(X+E)}{R} \right\rfloor$ or $\left\lfloor \frac{(X+E)}{R} \right\rfloor - 1$. If $\left\lfloor \frac{(X+E)}{R} \right\rfloor = 0$, then there is no error, and X is recovered correctly. Therefore, the value of $\left\lfloor \frac{(X+E)}{R} \right\rfloor = 0$ can be used as a syndrome, which determines if the results are correct.

Chapter 3. Methodology

This section introduces the methodology we use to design and evaluate the 2Lbp-RRNS model.

3.1 Research methodology

Here, we provide the main phases of an investigation and specific approaches used to identify, process, and analyze information about a topic:

- Study of the state-of-the-art solutions. The addressed problem requires understanding the state-of-the-art approaches to identify existed mechanisms, techniques, and open problems in the literature. We present the related works in Section 1.1. We describe the secret sharing schemes and error detection-correction codes in Chapter 2.

- Experimental evaluation. It consists of evaluating the performance of the state-of-the-art schemes and mechanisms by means of experimentation, adapting them to a data storage system in a multi-cloud environment using characteristics of real Cloud storage providers. We detail the experimentation in Chapter 4.

- Design of 2Lbp-RRNS model. The main objective is to develop a reliable data storage system. For this purpose, we design a two-level RRNS model with backpropagation and Hamming distance mechanisms that allows to improve reliability, redundancy, and reduce the overhead of data storage.

3.2 Multi-cloud environment approach

We consider a multi-cloud environment composed of a set of independent Cloud storage providers. Each of them stores a share without knowing of the other Cloud storages. On the one hand, since data are not stored on a single Cloud, it leads to the increased confidentiality. On the other hand, due to the redundancy property of RRNS, the system can recover the user information even if several storages are temporarily unavailable, corrupted, or lost. We use 11 providers from Table 1 of Section 2.1.1 based on their main characteristics and accessibility to implement the simulation framework.

To obtain the upload/download speeds, we perform a statistical analysis of Cloud storage access. We use file sizes from 10MB to 200MB to determine the data access rate. The shares are uploaded/downloaded to providers every hour during three days (Lopez-Falcon et al., 2019). In Table 4, we show the measurements for corresponding speeds.

We perform the experiments on an Intel® Xeon® CPU E5-1607 v3, with 16GB of memory. 64-bit OS, x64-based processor, and Chrome browser version 80.0.3987.149 with an Ethernet connection. The operating system is Windows 10.

Table 4. Speed access for seven Cloud storage providers

ID	Provider	Upload speed (MB/s)			Download speed (MB/s)		
		Min	Max	Ave	Min	Max	Ave
C_1	Google Drive	1.79	3.21	2.98	2.15	3.26	3.06
C_2	OneDrive	0.91	1.70	1.46	1.21	2.41	2.18
C_3	Dropbox	2.59	3.05	2.93	3.07	3.32	3.25
C_4	Box	1.91	3.26	2.55	2.01	3.20	2.62
C_5	Egnyte	1.24	1.93	1.70	2.17	2.36	2.30
C_6	ShareFile	0.11	0.65	0.51	0.72	0.76	0.75
C_7	SalesForce	0.52	0.73	0.64	0.68	0.72	0.71
C_8	Alibaba Cloud	2.32	3.14	2.73	2.54	3.18	2.86
C_9	Amazon Cloud Drive	0.70	1.86	1.28	2.49	3.09	2.79
C_{10}	Apple iCloud	2.05	3.45	2.75	2.01	2.98	2.49
C_{11}	Azure Storage	1.31	3.17	2.24	2.30	3.12	2.71

To evaluate the probability of the Cloud provider errors, we use the data from an analysis of downtime at IaaS public Cloud providers by CloudHarmony, (2015). It does not provide a holistic view of Cloud availability and complete picture of Cloud downtime statistics due to limited monitoring all Cloud provider services, all availability zones across multiple regions, etc. However, it serves as a valuable point of reliability analysis.

We use the definition of geometric probability to calculate the probability of failure for each Cloud. We use these results as the maximum probability of failure of each CSP. We assume that the minimum probabilities of failure is twice less. For example, OneDrive reported 11hrs 34 mins or 0.482 days of downtime, resulting in a maximum probability of failure of 0.00132043.

shows the results for eleven CS.

Table 5. Probability of failure for eleven Cloud storage providers

ID	Provider	Probability of failure		
		Min	Max	Ave
C_1	Google Drive	0.00072679	0.00145361	0.00109019
C_2	OneDrive	0.00066020	0.00132043	0.00099030
C_3	Dropbox	0.00097032	0.00194065	0.00145548
C_4	Box	0.00179699	0.00359402	0.00269549
C_5	Egnyte	0.00073249	0.00146503	0.00109874
C_6	ShareFile	0.00014269	0.00028542	0.00021404
C_7	SalesForce	0.00061739	0.00123480	0.00092609
C_8	Alibaba Cloud	0.00079897	0.00138793	0.00109345
C_9	Amazon Cloud Drive	0.00018227	0.00098241	0.00058234
C_{10}	Apple iCloud	0.00015664	0.00097632	0.00076648
C_{11}	Azure Storage	0.00039977	0.0087241	0.00063609

3.2.1 Allocation strategies to select Cloud storages

This section describes five allocations strategies proposed by Lopez-Falcon et al., (2019) to select the set of possible Cloud storages available or suitable to hold shares based on the system's requirements.

- *Random*: This strategy selects the first n Cloud storage accessible, arbitrary
- *BestUpload*: This strategy selects the first n available Cloud storage with the best upload speed access
- *BestDownload*: This strategy selects the first n available Cloud storage with the best download speed access
- *BestSecurity*: This strategy selects the first n available Cloud storage with the lowest probability of failure

- *AdaptiveSpeed*: Let $S_{u(n)}$ the set of n Cloud storage available with the best upload speed and $S_{d(n)}$ the set of download speed access. *AdaptiveSpeed* selects first the Cloud storage of $S_{u(n)}$, later the shares in $c \in \left(S_{u(n)} \backslash S_{d(n)}\right)$ are transferred to the set of Cloud storage $d \in \left(S_{d(n)} \backslash S_{u(n)}\right)$

- *AdaptiveSecurity*: Let $S_{u(n)}$ be the set of n available Cloud storage with the best upload speed and let $S_{pr(n)}$ be the set of Cloud storage with a lower probability of failure. First, *AdaptiveSecurity* selects the Cloud storage belonging to $S_{u(n)}$, later the shares in $c \in \left(S_{u(n)} \backslash S_{pr(n)}\right)$ are transferred to $d \in \left(S_{pr(n)} \backslash S_{u(n)}\right)$

3.3 Formal definition of the problem

Our objective is to develop a reliable distributed data storage system. For this purpose, we consider a set $C = \{c_1, c_2, \dots, c_m\}$. of m clouds. Each Cloud storage is described as a tuple $c_j = \{u_j, d_j, err_j\}$, where u_j is the upload speed, d_j download speed, and err_j the probability of failure, for $\forall_j = \{1, \dots, m\}$.

There are three necessary restrictions to our model:

- There must be at least two available Cloud storages (n ≥2).

- To recover the data, we need at least two shares (k≥2).

- The number of shares needed to retrieve the information must be less or equal to the total number of available Cloud storages (k≤n).

Figure 8 and Figure 9 show the basic model for data storage in a multi-cloud fashion.

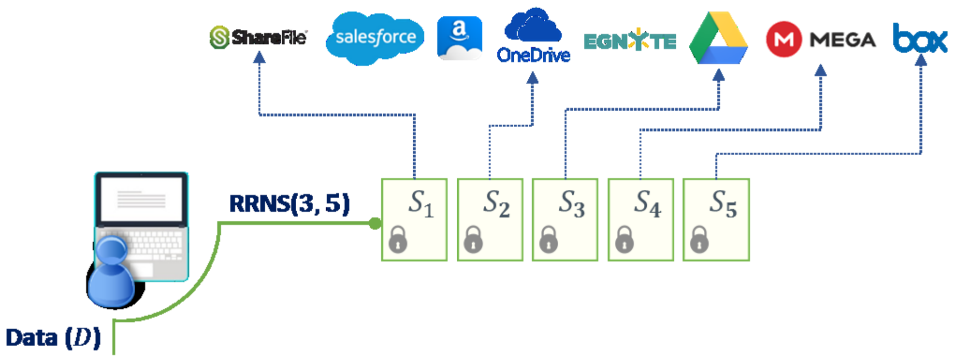


Figure 8. Encoding phase for a basic data storage model in a multi-cloud environment

Let assume a system with five storages and a (k, n)-threshold SSS with an access structure (3,5). In this scenario, we divide the original data D into five ($n = 5$) encoded shares $(s_1, s_2, s_3, s_4, s_5)$. Each share s_i is allocated to the Cloud selected from a pool of eight available clouds taking into account one of the allocation strategies listed in Section 3.2.1, see Figure 8.

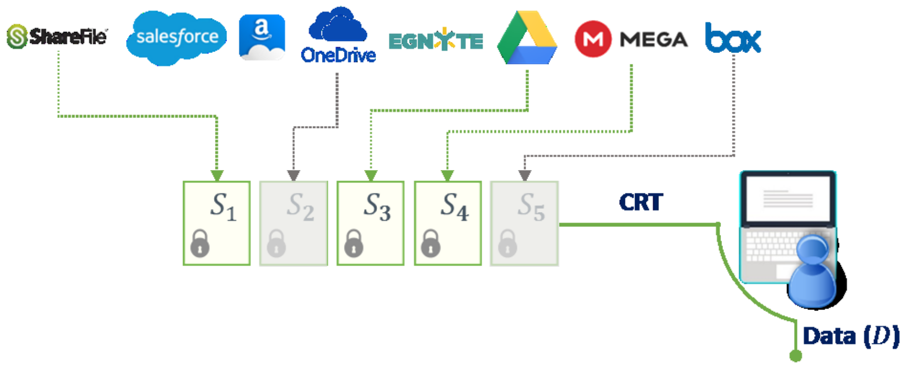


Figure 9. Decoding phase for a basic data storage model in a multi-cloud environment

To reconstruct D, we select three ($k = 3$) shares from the five storages. The selection of the k Clouds will depend on their availability at the moment. Figure 9 depicts that shares two and five are not used or unavailable. Finally, we obtain the original information using CRT or MRC methods.

3.4 Performance metrics

We consider several metrics to assess the quality of the strategies and the storage system: the number of detected and corrected errors, encoding/decoding and storing/extraction speeds, and data redundancy.

The number of detected and corrected errors are described in Section 3.5-3.7.

Redundancy (R) measures the ratio between the size of the original data (D) and the final size of the encrypted data (D_E). R is the extra information added in the coding phase due to the redundant moduli to ensure error detection and correction capability and is computed by:

$$R = \frac{size(D_E)}{size(D)} \tag{13}$$

Encoding speed (V_E) is the ratio between the original data size (D) and the total time that the algorithm takes to encode (encrypt) the shares. Where the encoding time (T_E) depends on the number of shares created.

$$V_E = \frac{size(D)}{T_E} \tag{14}$$

Decoding speed (V_D) is the ratio between the original data size (D) and the total time that the algorithm takes to decode (decrypt) the encoded data. Where the decoding time (T_D) depends on the number of shares that will be used to retrieve the data.

$$V_D = \frac{size(D)}{T_D} \tag{15}$$

Storing speed (V_s) is the ratio between the original data size (D) and the sum of storing time ($T_s = T_E + T_{up}$, where $T_{up} = \sum_{i=1}^{n} \frac{s_i}{u_i}$). V_s represents how fast information is divided, encrypted, and stored in the clouds.

$$V_s = \frac{size(D)}{T_s} \tag{16}$$

Extraction speed (V_{ex}) calculates how fast the k shares are downloaded and decoded to recover the original data. V_{ex} is computed as the ratio between original data length and the extraction time ($T_{ex} = T_D + T_{dow}$) which is the summation of decoding and downloading times, where $T_{dow} = \sum_{i=1}^{k} \frac{s_i}{d_i}$:

$$V_{ex} = \frac{size(D)}{T_{ex}} \tag{17}$$

We conducted a study with the average performance degradation proposed by Tsafrir et al., (2007) to choose the best strategy. First, we evaluate the performance degradation of each strategy under an arbitrary metric. We take the best value found for the strategy with the best performance as a reference. The equation for the performance degradation is:

$$(\gamma - 1) * 100 \text{ where } \gamma = \frac{strategy\ metric\ value}{best\ value\ found\ for\ the\ metric} \tag{18}$$

After this step, we rank the strategies based on the average values of the performance degradations. The approach with the minimum average value of the performance degradation is ranking in the first position. It is worth notice that we are trying to identify strategies that work reliably well in different settings; that is, we are trying to find a compromise that considers all of our test cases. The best strategy may not be the same for all scenarios.

3.5 RRNS Systems architectures

This section presents a detailed description of the RRNS architectures implemented and evaluated in the thesis.

3.5.1 One-Level RRNS (1L-RRNS)

This architecture splits the input file/data into a set of encoded (encrypted) shares with a smaller size. Then, we allocate a single share to each available Cloud storage. To reconstruct the original file, we need a specific number of shares or all of them. Figure 10 shows the basic design of a 1L-RRNS using three Cloud storage.

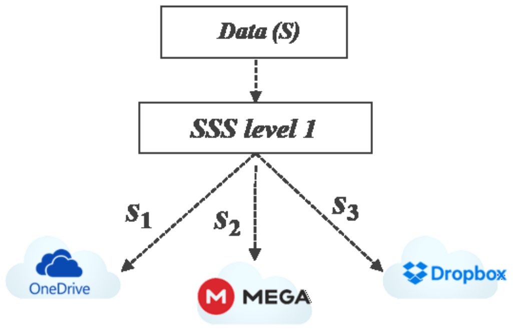


Figure 10. 1L-RRNS architecture

A 1L-RRNS model can improve resource sharing, improve performance reliability, and scalability. On the other hand, it may have lower security depending on the encoding/decoding mechanisms. It has multiple points of failure, significant complexity, unpredictability, and possible collusion of participants.

3.5.1.1 Basic process for encoding and decoding based on RRNS

In the following, we explain the basic process for encoding and decoding the data based on CRT. First, we select an access structure (k, n) and a set of available storages using a configurable algorithm proposed by García-Hernández et al., (2020). Then, we calculate the dynamic range $P = \prod_{i=1}^{k} p_i$. The input file is read by blocks of bytes, calculated using the dynamic range $(LenBlock(P))$.

We represent each read block by its residue representation by $|block_1|_{p_i}$ for $1 \le i \le n$. Next, we store the residue representation in an output file; after this, we read the next block, and we do the same process until we encode the whole data in n shares. Figure 11 depicts the encoding process.

For recovery, we download k shares from the available Cloud storage. We read from each share, blocks of size $LenBlock(P)$, and recover the original information applying CRT.

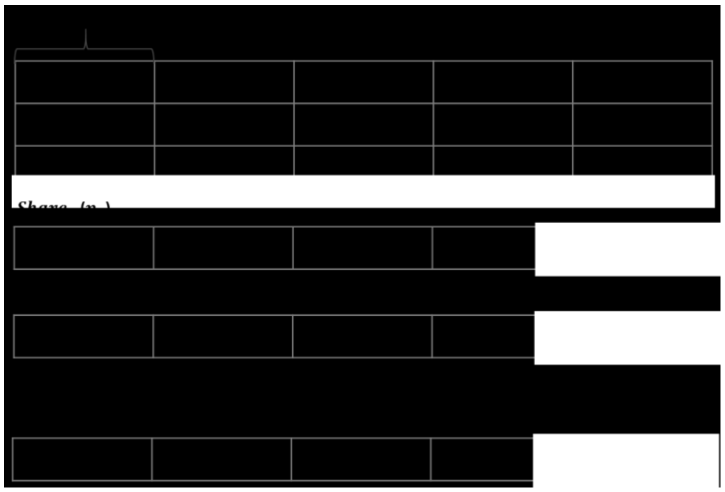


Figure 11. The encoding process for an RRNS system

Example:

Coding and decoding using (k, n)-Mignotte threshold scheme:

Suppose an access structure (2,4), and input data $D = 4056_{10} = 111111011000_2$. Suppose the RRNS moduli set $p_1 = 11_{10} = 1011_2, p_2 = 13_{10} = 1101_2, p_3 = 17_{10} = 10001_2$, and $p_4 = 19_{10} = 10011_2$, where 17 and 19 are the redundant moduli.

Encoding

1. We need to determine the block length based on the dynamic range $P = 11 * 13 = 143_{10}$

   We calculate the number of bits to represent P, since $19 < 2^7 < 143$, hence we divide D into blocks of 7 bits.

   $$D = 4056_{10} = \underbrace{11111_2}_{block_2}\underbrace{1011000_2}_{block_1}$$

2. We obtain the residue representation of each block.

   $$block_1 = 88_{10} = 101100_2 \xrightarrow{RRNS} (0_{10}, 10_{10}, 3_{10}, 12_{10})$$

   $$block_2 = 31_{10} = 11111_2 \xrightarrow{RRNS} (9_{10}, 5_{10}, 14_{10}, 12_{10})$$

3. We create the n shares.

$$S_1 = \underbrace{0000}_{block_{1,1}} \underbrace{1001}_{block_{2,1}} = 9_{10}$$

$$S_2 = \underbrace{1010}_{block_{1,2}} \underbrace{0101}_{block_{2,2}} = 165_{10}$$

$$S_3 = \underbrace{0011}_{block_{1,3}} \underbrace{1110}_{block_{2,3}} = 62_{10}$$

$$S_4 = \underbrace{1100}_{block_{1,4}} \underbrace{1100}_{block_{2,4}} = 204_{10}$$

4. We send each share S_i to $n = 4$ available Cloud storages taking into account their characteristics.

Decoding

1. We retrieve $k = 2$ shares (S_i) from the Clouds. Suppose that Cloud 2 and Cloud 3 are not available, so we only have shares from Cloud 1 and Cloud 4.

$S_1 = \underbrace{0000}_{block_{1,1}} \underbrace{1001}_{block_{2,1}} = 9_{10}$ and $S_4 = \underbrace{1100}_{block_{1,4}} \underbrace{1100}_{block_{2,4}} = 204_{10}$

2. The moduli we need are $p_1 = 11_{10} = 1011_2$ and $p_4 = 19_{10} = 10011_2$.

3. We use classic CRT to recover D.

$P = 11 * 19 = 209_{10}$, $P_1 = \frac{209}{11} = 19$, $P_4 = \frac{209}{19} = 11$

The multiplicative inverses are $\left|P_1^{-1}\right|_{p_1} = |19^{-1}|_{11} = 7$, $|P_4^{-1}|_{p_4} = |11^{-1}|_{19} = 7$

$$CRT_1 = \left|block_{1,1} \cdot \left|P_1^{-1}\right|_{p_1} \cdot P_1 + lock_{1,4} \cdot |P_4^{-1}|_{p_4} \cdot P_4\right|_P = |0 \cdot 7 \cdot 19 + 12 \cdot 7 \cdot 11|_{209} = 88_{10}$$
$$= 1011000_2$$

$$CRT_2 = \left|block_{2,1} \cdot \left|P_1^{-1}\right|_{p_1} \cdot P_1 + lock_{2,4} \cdot |P_4^{-1}|_{p_4} \cdot P_4\right|_P = |9 \cdot 7 \cdot 19 + 12 \cdot 7 \cdot 11|_{209} = 31_{10}$$
$$= 11111_2$$

4. We reconstruct $D = \underbrace{11111}_{CRT_2}\underbrace{1011000}_{CRT_1} = 11111101100_2 = 4056_{10}$

If more than two shares were unavailable, the system could not recover the data since there are not enough redundant moduli.

3.5.2 Two-Level RRNS (2L-RRNS)

This architecture divides the input file into smaller encoded (encrypted) shares as in 1L-RRNS. Then, we take each share, and we split it again, creating smaller ones. Finally, we distribute each smaller share according to the available Cloud storage characteristics. To recover the data, we first reconstruct the set of k_2 shares for the second level, and from these, we recover k_1 shares of the first level, and reconstruct the original data, see Figure 12.

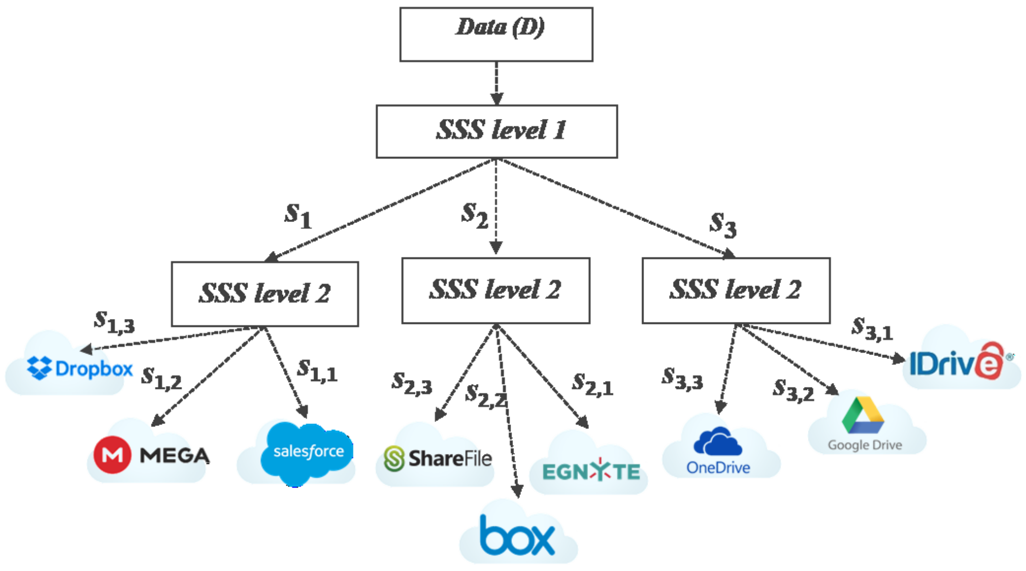


Figure 12. 2L-RRNS architecture

In a 1L-RRNS architecture, to increase the reliability, it is necessary to use a large number of small moduli. For small moduli, converting numbers from RRNS to a binary system may be more complex. On the other hand, large moduli may lead to security imbalance since the data may not be encoded, revealing partial information of the original data. One of the significant advantages of a 2L-RRNS model is the simple selection of moduli set to guarantee the required dynamic range. Besides, this model improves coding/decoding performance, computational security, reliability reducing data loss risks.

3.6 Data storage systems models

We evaluate the different schemes using two models: Equal-Share and Short-Share. The first one is the most basic system model used in most of the related works. In the second model, each Cloud storage holds more than one short share.

3.6.1 Equal-Share (ES-RRNS)

Equal-Share (ES-RRNS) is the basic model depicted in Figure 13. In ES-RRNS, the shares have the same length. The number of shares is equal to the number of Cloud storages established by the (k, n) parameters of the system.

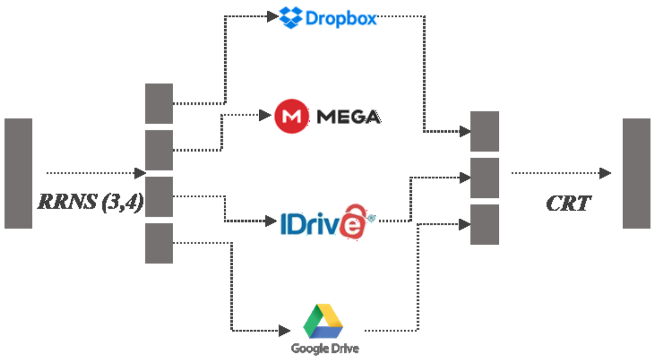


Figure 13. Equal-Share (ES-RRNS) system model

3.6.2 Short-Share (SS-WA-RRNS)

Short-Share (SS-WA-RRNS) model is depicted in Figure 14. WA-RRNS is a weighted-RRNS scheme. We use two versions: WA-RRNS based on CRT (Tchernykh et al., 2018) and WA-MRC-RRNS based on MRC (A.

Tchernykh et al., 2020). WA-RRNS splits the secret into a set of short encoded shares. This approach reduces the probability of information loss at the expense of reduced coding and decoding speeds.

The weighted threshold access scheme WA-RRNS is noted as (n_v, k, N) where N is the total number of clouds ($N \geq 2$). $n_v = (n_1, n_2, \dots, n_N)$ is a sequence of positive integers, where n_v defines the number of shares sent to each Cloud. And, k is a positive integer such that $2 \leq k \leq \sum_{i=1}^{N} n_i$. The parameters $n_1, n_2, \dots, n_N$ will be referred to as the weights. If $n_i \geq k$ exists, then i-th cloud can restore the encoded data and violate the confidentiality rules. A weighted system where $n_v = (k-1, k-1, \dots, k-1)$ provides the maximum reliability considering the limitation $n_i < k$.

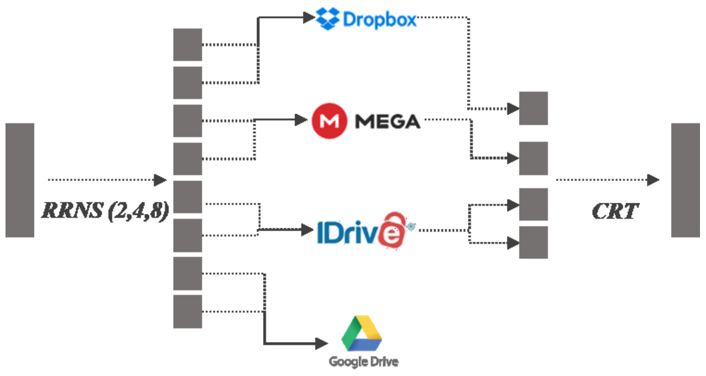


Figure 14. Short-Share (SS-WA-RRNS) system model

3.7 2Lbp-RRNS: Two-Levels RRNS with Backpropagation and Hamming distance mechanisms

In this thesis, we focus on 2L-RRNS architecture. We present an algorithm capable of improving the system's reliability. We do not use a special type of moduli set to enhance the reliability of the system. This model uses a backpropagation mechanism and Hamming Distance techniques to detect and correct errors.

3.7.1 2L-RRNS formal definition

We describe the RRNS parameters for a two-level RRNS architecture using the notation presented at the beginning of this document (see, Basic Notation). We give the RRNS parameters that describe the first level; then, the RRNS parameters involve the second level.

For the first level, we have a pairwise coprime moduli set $\{p_{1,1}, \dots, p_{1,2}, p_{1,n_1}\}$, where $n_1 = k_1 + r_1$. We define the legal (dynamic) range by $P = \prod_{i=1}^{k_1} p_{1,i}$. A data S represented in the Binary-Weighted Number System (BNS) has a residue representation by the tuple $S \xrightarrow{RNS} (S_1, S_2, \dots, S_n)$ where $S \in [0, P)$, and $S_i = |S|_{p_{1,i}}$ $i = \overline{1, n_1}$. A 1L-RRNS with an access structure (k_1, n_1) with r_1 redundant (control) moduli, then, according to 1L-RRNS properties, the system can detect $r_1 = n_1 - k_1$ and correct $\lfloor r_1/2 \rfloor$ errors. 2L-RRNS uses the Projection method (see, Section 2.4.2.1) as an error correction code like a classical 1L-RRNS, where the number of calculated projections grows exponentially depending on r_1, hence 2L-RRNS's capability of correction has the same limitations and restrictions.

For the second level, we have the moduli set $\{p_{2,i,1}, \dots, p_{2,i,2}, p_{2,i,n_{2,i}}\}$, where $i = \overline{1, n_1}$. Thus, each S_i of the first level, is transformed into the set of residuals by $S_{i,j} = |S_i|_{p_{2,i,j}}$, see Figure 15. The dynamic range of the second level is given by $M_i = \prod_{j=1}^{k_{2,i}} p_{2,i,j} \geq p_{1,i}$. S_i satisfies the condition $S_i < p_{1,i}$, for all $i = \overline{1, n_1}$.

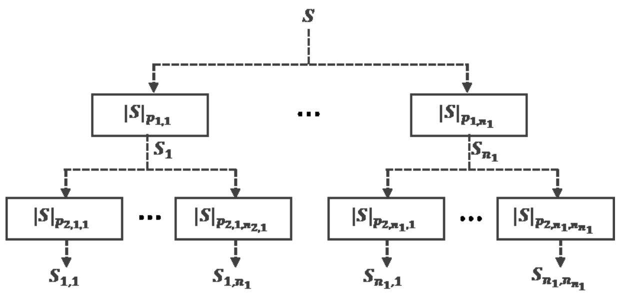


Figure 15. 2L RRNS encoding

From the CRT, it follows that for a one-to-one mapping between $S_i \in [0,\ p_{1,i})$ and $\widetilde{S_i} = (S_{i,1}, S_{i,2}, \dots, S_{i,n_{2,i}})$, it is necessary and sufficient that $M_i \geq p_{1,i}$ for each $i = \overline{1, n_1}$.

Multi-Operand Modulo Addition (MOMA) is an algorithmic primitive that accepts n_1 operands $S_1, S_2, \dots, S_{n_1}$, with $0 \leq S_i < p_{1,i}$ for each $i = \overline{1, n_1}$ and computes the residue of their sum taken modulo P. That is, it calculates original data S, as $S = |w_i S_1 + w_2 S_2 + \dots + w_{n_1} S_{n_1}|_P$, where $w_i = P_i \cdot |P_i^{-1}|_{p_{1,i}}$ and $P_i = P / p_{1,i}$, for all $i = \overline{1, n_1}$ (Piestrak, 1994).

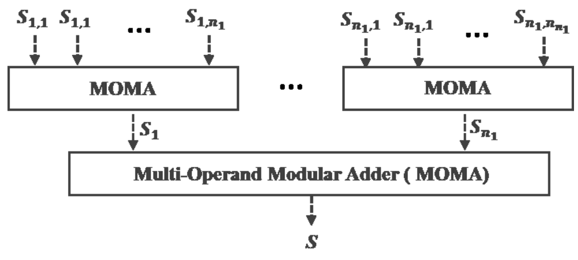

$S_{1,1}$ $S_{1,1}$ … S_{1,n_1}
MOMA
…
$S_{n_1,1}$ $S_{n_1,1}$ … $S_{n_1,n_{n_1}}$
MOMA
S_1
S_{n_1}
Multi-Operand Modular Adder (MOMA)
S

Figure 16. 2L-RRNS decoding

We design 2Lbp-RRNS as an extension of the classical 2L-RRNS. We discuss how the reliability and performance of the system depend on the de access structure (k, n) on each level. In the following sections, we present error correction details by showing the idea behind the state-of-the-art 2L-RRNS algorithm proposed by Barati et al., (2008) and our 2Lbp-RRNS solution.

3.7.2 2L-RRNS error corrections

Let's recall that a 2L-RRNS is an architecture in which the shares of one level are the next level's entries, obtaining a final set of shares with smaller sizes than a 1L-RRNS approach, see Figure 12. Suppose we have a set of n_1 shares $(S_1, S_2, \dots, S_{n_1})$ in 2L-RRNS, we can calculate the value of each S_i, if and only if, the number of errors is less than or equal to $\left\lfloor \frac{r_{2,i}}{2} \right\rfloor$. In all other cases, we can say that S_i is incorrect (similar to 1L-RRNS) but we cannot recover it.

Algorithm 1, named "2L-RRNS error correction," applies the general Projection method (see, Section 2.4.2.1) of the 1L-RRNS on each level. We do not use the syndrome method (see, Section 2.4.2.2). The amount of memory required by the syndrome method in 1L-RRNS increases exponentially, depending on the number of errors to be corrected (N. Chervyakov et al., 2019). Thus, to store in memory a table of constants for the second level requires increasing the memory by n_1 times, which is not reasonable.

The function *CRTtoBin* converts numbers from RRNS to a BNS using CRT. The *ProRRNS* function calculates the projection value in RRNS; this function allows the system's error correction capability.

Barati et al., (2008) show the next Theorem 2, to demonstrate the number of detected N_D^{2L} and corrected N_E^{2L} errors of 2L-RRNS.

Algorithm 1. 2L-RRNS error correction

```
Input: $(k_1, n_1)$, $(k_{2,1}, n_{2,1})$, …, $(k_{2,n_1}, n_{2,n_1})$
        $S_1 \xrightarrow{RNS} (S_{1,1}, S_{1,2}, \dots, S_{1,n_{2,1}})$,
        $S_2 \xrightarrow{RNS} (S_{2,1}, S_{2,2}, \dots, S_{2,n_{2,2}})$,
        … ,
        $S_{n_1} \xrightarrow{RNS} (S_{n_1,1}, S_{n_1,2}, \dots, S_{n_1,,n_{2,n_1}})$,
        $(p_{1,1}, \dots, p_{1,n_1}), (p_{2,1,1}, \dots, p_{2,1,n_{2,1}}), \dots, (p_{2,n_1,1}, \dots, p_{2,n_1,n_{2,n_1}})$.
Output: $S, flag, (S_1, S_2, \dots, S_{n_1})$.
        $flag = 0$, if there are no errors, $flag = 1$ if errors are detected and corrected, $flag = -1$, if errors are
        detected but not corrected.
Begin
1. $flag = 0$
2. For $i = 1$ to $n_1$ do:
   2.1. $S_i' = CRTtoBin((S_{i,1}, S_{i,2}, \dots, S_{i,n_{2,i}}))$
   2.2. If $S_i' \geq P_{2,i}$ then:
        2.2.1. $temp = ProRRNS\left(S_i',\ \left(p_{2,i,1}, p_{2,i,2}, \dots, p_{2,i,n_{2,i}}\right),\ k_{2,i},\ val\right)$
        2.2.2. $flag = flag + temp$;
        2.2.3. If $temp == 0$ then $S_i = val$ else $S_i = -1$
3. If $flag == 0$ then:
  3.1. $S = CRTtoBin((S_1, S_2, \dots, S_{n_1}))$
4. else
  4.1 If $flag \leq \left\lfloor \frac{n_1 - k_1}{2} \right\rfloor$ then:
     4.1.1 $flag = 1$
     4.1.2. $S = CRTtoBin((S_1, S_2, \dots, S_{n_1}))$
  4.2. else $flag = -1$
5. return $S, flag, (S_1, S_2, \dots, S_{n_1})$
End
```

Theorem 1. *2L-RRNS can detect N_D^{2L} errors and correct N_E^{2L} errors, where*

$$\mathrm{N}_D^{2L} = \sum_{i=1}^{n_1} \left(n_{2,i} - k_{2,i}\right), \qquad \mathrm{N}_E^{2L} = \sum_{i=1}^{n_1} \left\lfloor \frac{n_{2,i} - k_{2,i}}{2} \right\rfloor \tag{19}$$

Proof

Since the amount of minimum Hamming distance in a multi-level redundant residue system, is equal to $\sum_{i=1}^{n_1}(r_i + 1)$, so:

If in each second level representation of S^j which is assumed for the first level of $i-th$ moduli ($i \in \overline{1,n_1}$), up to maximum of $(n_{2,i}, k_{2,i})$ errors occur, then the 2L-RRNS system can detect $\mathrm{N}_D^{2L} = \sum_{i=1}^{n_1}(n_{2,i} - k_{2,i})$ errors.

If in each second level representation of S^j which is assumed for the first level of $i-th$ moduli ($i \in \overline{1,n_1}$), up to maximum of $\left\lfloor \frac{(n_{2,i}, k_{2,i})}{2} \right\rfloor$ errors occur, then the 2L-RRNS system can correct $\mathrm{N}_E^{2L} = \sum_{i=1}^{n_1}\left\lfloor \frac{n_{2,i}-k_{2,i}}{2} \right\rfloor$ errors.The theorem is proved.

To better understand the 2Lbp-RRNS properties, let us first consider special cases of 1L-RRNS and 2L-RRNS when we know the localization of errors. We use special subscripts for detection Dl and correction El cases with error localization.

Lemma 2. For (k_1, n_1) *1L-RRNS, if we know the localization of* k_1 *correct* S_i*. 1L-RRNS can restore* S*.*

Proof

Without loss of generality, let the correct values be $S_{i_1}, S_{i_2}, \dots, S_{i_{k_1}}$, then using the CRT, *1L-RRNS* can restore S using the formula: $S = \left| w_1 S_{i_1} + w_2 S_{i_2} + \cdots + w_{k_1} S_{i_{k_1}} \right|_{P_I}$, where $P_I = \prod_j^{k_1} p_{i_j}$ and $w_j = \frac{P_I}{p_{i_j}} \cdot \left| \frac{p_{i_j}}{P_I} \right|_{p_{i_j}}$ The lemma is proved.

Corollary 1. For (k_1, n_1) *1L-RRNS, if we know the localization of* k_1 *correct* S_i*, 1L-RRNS can correct* $\mathrm{N}_{El}^{1L} \le r_1 = n_1 - k_1$ *errors.*

Proof

a) When we know $S_{i_1}, S_{i_2}, \dots, S_{i_{k_1}}$ with no errors, then the condition of Lemma 2 is satisfied. Therefore, we can restore the result by correcting $r_1 = n_1 - k_1$ errors.

b) When there exist $k_1 - 1$ values with no errors, given that (k_1, n_1) is an access structure of a threshold-SSS. Hence, we cannot restore the actual value of S.

Therefore, if there is an algorithm that can determine which of S_i is correct, we can correct no more than $r_1 = n_1 - k_1$ errors. The corollary is proved.

Lemma 3. *If there is an algorithm that can determine which of $S_{i,j}$ is true, then the 2L-RRNS can correct $\mathrm{N}_{El}^{2L} \leq \sum_{i=1}^{n_1} n_{2,i} - \sum_{i=1}^{k_1} k_{2,i}$ errors.*

Proof

Without loss of generality, we assume that $k_{2,1} \leq k_{2,2} \leq \cdots \leq k_{2,n_1}$. Two cases may take place:

Case 1. Let us assume that $r_{2,i}$ errors happened in the representation of S in 2L-RRNS, $\widetilde{S^i}$ for all $i \in \overline{1,k_1}$, for a total number of errors $\sum_{i=1}^{k_1} r_{2,i} = \sum_{i=1}^{k_1} (n_{2,i} - k_{2,i})$, and $n_{2,i}$ errors for all $i \in \overline{k_1 + 1, n_1}$, with total errors equal to $\sum_{i=k_1+1}^{n_1} n_{2,i}$. Following Lemma 2, we can restore the actual value of S_i for all $i \in \overline{1,k_1}$. Therefore, the condition of Lemma 2 is satisfied, and it is possible to restore the true value of S correcting N_{El}^{2L} errors.

$$\mathrm{N}_{El}^{2L} \leq \sum_{i=1}^{k_1} \left(n_{2,i} - k_{2,i}\right) + \sum_{i=k_1+1}^{n_1} n_{2,i} = \sum_{i=1}^{k_1} n_{2,i} - \sum_{i=1}^{k_1} k_{2,i} + \sum_{i=k_1+1}^{n_1} n_{2,i} = \sum_{i=1}^{n_1} n_{2,i} - \sum_{i=1}^{k_1} k_{2,i} \qquad (20)$$

Case 2. If we add one error more to $\widetilde{S^i}$ (without loss of generality, we will consider that the error is added to S_j representations), for $i \in \overline{1,k_1}$, then, according to Corollary 1, we cannot restore the actual value of S_i for all $i \in \{k_1 + 1, k_1 + 2, \ldots, n_1\} \cup \{j\}$, and according to Lemma 2, we can restore the true value of S_i for all $i \in \{1,2, \ldots, k_1\} \backslash \{j\}$, therefore, we cannot restore the real value of S.

From the first and second cases, it follows that the number of errors that 2L-RRNS can correct is:

$$\mathrm{N}_{El}^{2L} \leq \sum_{i=1}^{n_1} n_{2,i} - \sum_{i=1}^{k_1} k_{2,i} \qquad (21)$$

Lemma 3 is proved.

3.7.3 Algorithm 2Lbp-RRNS with backpropagation and Hamming distance

Next, we describe the algorithm to implement our 2Lbp-RRNS; we also provide the number of detected and corrected errors of 2Lbp-RRNS, using the backpropagation and Hamming distance mechanisms. Let us discuss Algorithm 2.

Let us consider the example when the access structure (2,3) is on the first and the second level. Thus, $k_1 = k_{2,1} = k_{2,2} = k_{2,3} = 2, n_1 = n_{2,1} = n_{2,2} = n_{2,3} = 3$.

On the first level, we have a tuple with three elements (S_1, S_2, S_3). On the second level, we have three tuples, each with three elements $(S_{1,1}, S_{1,2}, S_{1,3})$, $(S_{2,1}, S_{2,2}, S_{2,3})$, and $(S_{3,1}, S_{3,2}, S_{3,3})$. We suppose errors in $S_{1,3}, S_{2,3}, S_{3,3}$. In this case, traditional 2L-RRNS detects errors but cannot correct them because it does not know the position of errors.

Following the Algorithm2, in the first step, 2Lbp-RRNS applies Algorithm 1. Based on the resulting shares $S_{1,1}, S_{1,2} \ldots . S_{n_1,n_{2,n_1}}$, the algorithm choose a subset of $k_{2,i}$ shares for each $i = \overline{1, n_1}$. Next, we calculate three possible tuples S_i^1, S_i^2, and S_i^3 to recover each S_i.

So to recover S_1 we have tuples S_1^1, S_1^2, and S_1^3

$S_1^1 \overset{RNS}{\longleftarrow} \widetilde{S_1^1} = (S_{1,1}, S_{1,2})$, $S_1^2 \overset{RNS}{\longleftarrow} \widetilde{S_1^2} = (S_{1,1}, S_{1,3})$, and $S_1^3 \overset{RNS}{\longleftarrow} \widetilde{S_1^3} = (S_{1,2}, S_{1,3})$.

We do the same for S_2 denoting possible tuples as:

$S_2^1 \overset{RNS}{\longleftarrow} \widetilde{S_2^1} = (S_{2,1}, S_{2,2})$, $S_2^2 \overset{RNS}{\longleftarrow} \widetilde{S_2^2} = (S_{2,1}, S_{2,3})$, and $S_2^3 \overset{RNS}{\longleftarrow} \widetilde{S_2^3} = (S_{2,2}, S_{2,3})$.

Then, Algorithm 2 tries to recover S, we analyze nine possible tuples of the first level denoted as:

$S^1 \overset{RNS}{\longleftarrow} \widetilde{S^1} = (S_1^1, S_2^1)$, $S^2 \overset{RNS}{\longleftarrow} \widetilde{S^2} = (S_1^1, S_2^2)$, $S^3 \overset{RNS}{\longleftarrow} \widetilde{S^3} = (S_1^1, S_2^3)$,

$S^4 \overset{RNS}{\longleftarrow} \widetilde{S^4} = (S_1^2, S_2^1)$, $S^5 \overset{RNS}{\longleftarrow} \widetilde{S^5} = (S_1^2, S_2^2)$, $S^6 \overset{RNS}{\longleftarrow} \widetilde{S^6} = (S_1^2, S_2^3)$,

$S^7 \overset{RNS}{\longleftarrow} \widetilde{S^7} = (S_1^3, S_2^1)$, $S^8 \overset{RNS}{\longleftarrow} \widetilde{S^8} = (S_1^3, S_2^2)$, $S^9 \overset{RNS}{\longleftarrow} \widetilde{S^9} = (S_1^3, S_2^3)$.

Based on the backpropagation concept, we represent each of nine S^j values back to 2L-RRNS representation denoting $\widetilde{S^j}$.

Algorithm 2. 2Lbp-RRNS error corrections

```
Input: $(k_1, n_1)$, $(k_{2,1}, n_{2,1})$, …, $(k_{2,n_1}, n_{2,n_1})$
        $S_1 \xrightarrow{RNS} (S_{1,1}, S_{1,2}, \dots, S_{1,n_{2,1}})$,
        $S_2 \xrightarrow{RNS} (S_{2,1}, S_{2,2}, \dots, S_{2,n_{2,2}})$,
        … ,
        $S_{n_1} \xrightarrow{RNS} (S_{n_1,1}, S_{n_1,2}, \dots, S_{n_1,,n_{2,n_1}})$,
        $(p_{1,1}, \dots, p_{1,n_1}), (p_{2,1,1}, \dots, p_{2,1,n_{2,1}}), \dots, (p_{2,n_1,1}, \dots, p_{2,n_1,n_{2,n_1}})$.
Output: $S, flag$
Begin
Step 1. 2L-RRNS calculates S and flag. If flag≠-1, S is recovered, otherwise S is not recovered.
Step 2. Based on $S_{1,1}, S_{1,2} \dots . S_{n_1,,n_{2,n_1}}$, choosing an (unordered) subset of $k_{2,i}$ elements from a fixed set of $n_{2,i}$
        elements, we calculate possible values $S_i^l$ for each of $S_i$.
Step 3. Choosing an (unordered) subset of $k_1$ elements $S_i^l$ from a fixed set of $n_1$ elements, we calculate possible
        values $S_i^l$ and restore $S^j$ by the function CRTtoBin.
Step 4. Using the backpropagation concept, we encode each $S^j$ to 2L-RRNS representation $\widetilde{S^j}$ and compute the
        HD between $\widetilde{S^j}$ and $\bar{S}$.
Step 5. We choose $\widetilde{S^j}$ for which the HD is minimal. If minimal HD between $\widetilde{S^j}$ and $\bar{S}$ is more than $N_E^{2Lbp} =$
        $\sum_{i=1}^{n_1} n_{2,i} - \sum_{i=1}^{k_1} k_{2,i} - 1$, then return $flag = -1$; otherwise, $S = S^j$ and flag $= 1$.
End
```

For each resulting $\widetilde{S^j}$, we calculate the Hamming distance between $\bar{S}$ and $\widetilde{S^j}$, for all $j = \overline{1,9}$. In our example, the Hamming Distance equals three for all $j = \overline{2,9}$, and equals to two between $\bar{S}$ and $\widetilde{S^1}$. We note that k=2, hence, $S = S^1$. We can restore the data.

3.7.3.1 2Lbp-RRNS error correction

Now, let us calculate the number of errors detected and corrected by 2Lbp-RRNS in the general case. This method's basic idea is in the backpropagation of the restored variant that the algorithm cannot prove as correct or incorrect. This restored variant is encoded, so the algorithm moves the data in the direction reverse to restoring. This new encoded variant $\widetilde{S^j}$ is compared with the initially encoded value $\bar{S}$ by calculating the HD. If HD is less than a given threshold, $\widetilde{S^j}$ is considered as a correct restored S. Thus, the error correction of 2Lbp-RRNS includes two additional processes: 2L-RRNS encoding and HD calculation.

The number of possible variants $\widetilde{S^{J}}$ depends on the number of errors in $\overline{S}$. Due to a large number of possible combinations, the time can grow significantly. On the one hand, backpropagation increases the error detection and correction algorithm's computational complexity, similar to the base extension (Watson & Hastings, 1966). On the other hand, it allows us to identify and recover more errors.

First, we discuss the HD properties for the error localization in 2L-RRNS in more detail. Let we have two 2L-RRNS representations of S: without errors $\tilde{S}$ and with errors $\bar{S}$. We can establish the following property 1:

Property 1. *If* $HD(\tilde{S},\bar{S}) = 0$ *then* $\bar{S}$ *does not contain errors, i.e.* $\tilde{S} = \bar{S}$.

Proof

Evidence from the contrary. Assume that $\bar{S}$ contains errors and $HD(\tilde{S},\bar{S}) = 0$. Since $\bar{S}$ contains errors, there exists a representation $\bar{S} \xleftarrow{2L-RRNS} S + E$, where $0 < E < P$, therefore:

$$\bar{S} = \left(\left(S'_{1,1}, \dots, S'_{1,n_{2,1}}\right), \dots, \left(\left(S'_{n_1,1}, \dots, S'_{n_1,n_{2,n_1}}\right)\right)\right),$$

$$\bar{E} = \left(\left(E'_{1,1}, \dots, E'_{1,n_{2,1}}\right), \dots, \left(\left(E'_{n_1,1}, \dots, E'_{n_1,n_{2,n_1}}\right)\right)\right),$$

$$\tilde{S} = \left(\left(S_{1,1}, \dots, S_{1,n_{2,1}}\right), \dots, \left(\left(S_{n_1,1}, \dots, S_{n_1,n_{2,n_1}}\right)\right)\right),$$

where for all i,j: $S'_{i,j} = S_{i,j} + E_{i,j}$.

Considering that $0 < E < P$ then there exists at least one pair (i,j), such that $E_{i,j} \neq 0$, therefore there is at least one value $S_{i,j} \neq S'_{i,j}$, then $HD(\tilde{S},\bar{S}) > 0$. Thus, we have a contradiction. Therefore, if $HD(\tilde{S},\bar{S}) = 0$ then $\bar{S}$ does not contain errors. The property is proved.

Corollary 2. *The number of errors that has 2L-RRNS representation of* $\bar{S}$ *is equal to* $HD(\tilde{S},\bar{S})$.

Property 2. Let $S^1 \neq S^2 \neq \cdots \neq S^t$, for which $HD(\widetilde{S^1},\bar{S}) = HD(\widetilde{S^2},\bar{S}) = \cdots = HD(\widetilde{S^t},\bar{S})$, then the number of errors in each of the representations $\widetilde{S^j}$ is equal.

Proof

Since the number of errors in the 2L-RRNS representation of S^j by the Corollary 2 is determined by $HD(\widetilde{S^j},\bar{S})$, from the condition $HD(\widetilde{S^1},\bar{S}) = HD(\widetilde{S^2},\bar{S}) = \cdots = HD(\widetilde{S^t},\bar{S})$, it follows that the number of errors in each $\widetilde{S^j}$ is equal, for all $j = \overline{1,t}$. The property is proved.

Corollary 3. Let $S^1 \neq S^2 \neq \cdots \neq S^t$, for which $HD(\widetilde{S^1},\bar{S}) < HD(\widetilde{S^2},\bar{S}) < \cdots < HD(\widetilde{S^t},\bar{S})$, then the S^1 representation in 2L-RRNS contains the least errors.

It follows from Corollary 3 and Property 2. If there are at least two S^j values such that $S^1 \neq S^2$, $HD(\widetilde{S^1},\bar{S}) = HD(\widetilde{S^2},\bar{S}) = d$ and $d \leq HD(\widetilde{S^j},\bar{S})$ for all $j = \overline{1,t}$, then it is impossible to correct S since we cannot determine which value from two S^1, S^2 is true.

Theorem 2. *2Lbp-RRNS can detect N_D^{2Lbp} errors and correct N_E^{2Lbp} errors, where*

$$\mathrm{N}_D^{2Lbp} = \sum_{i=1}^{n_1} n_{2,i} - \sum_{i=1}^{k_1} k_{2,i},$$

$$\mathrm{N}_E^{2Lbp} \leq \sum_{i=1}^{n_1} n_{2,i} - \sum_{i=1}^{k_1} k_{2,i} - 1 \qquad (22)$$

Proof

From Corollary 2, it follows that the maximum number of errors that we can determine using Hamming Distance is equal to the maximum number of errors that can be corrected if we know them, therefore

$$\mathrm{N}_D^{2Lbp} = \sum_{i=1}^{n_1} n_{2,i} - \sum_{i=1}^{k_1} k_{2,i}. \tag{23}$$

To estimate the number of correctable errors of 2Lbp-RRNS, we consider their upper bound. As in traditional threshold 2L-RRNS, no more than $\lfloor r_{2,i}/2 \rfloor$ errors are localized in each $\widetilde{S^i}$. Hence, there are $k_1 + \lceil r_{2,i}/2 \rceil$ values of S_i, the correctness of which can be confirmed. Hence, we can correct the actual value.

Without loss of generality, we assume that in each of the representations in 1L-RRNS $S_{i_q} \in \{S_{i_1}, S_{i_2}, \dots, S_{i_l}\}$ contains no more than r_{2,i_q} errors, where $l \geq k_1$. We denote $I = \{i_1, \dots, i_l\}$ and there exists $u \notin I$ for which the representation in 1L-RRNS S_u contains less than $n_{2,u}$ errors, then using Corollary 3, we can restore S based on the backpropagation mechanism. Backpropagation encodes each S^j candidate of S back to RRNS representation denoting $\widetilde{S^j}$.

Consider the number of errors that the upper bound case can correct. The maximum number of errors is less than or equal to

$$\mathrm{N}_E^{2Lbp} \leq \sum_{i=1}^{k_1} \left(n_{2,i} - k_{2,i}\right) + \sum_{i=k_1+1}^{n_1} n_{2,i} - 1 = \sum_{i=1}^{n_1} n_{2,i} - \sum_{i=1}^{k_1} k_{2,i} - 1. \tag{24}$$

The theorem is proved.

We use Theorem 2 and Theorem 3 to calculate the capability of the 2LRRNS and 2Lbp-RRNS respectively. We use the access structures described in

Table **6** to explore all possible scenarios. Where $(k_1, n_1) = (k_{2,i}, n_{2,i})$, each Cloud storage has the same number of shares with the same threshold.

Figure 17 and Figure 18 show the number of detected and corrected errors. We observe that 2Lbp-RRNS can detect and correct more errors than 2L-RRNS for all test cases. For the give results, 2Lbp-RRNS can detect 1.58x and correct 3.37x times more errors than 2L-RRNS, on average.

Table 6. Scheme settings for evaluation of error detection and correction

id	k_1	n_1	$(k_{2,1}, n_{2,1}) - \cdots - (k_{2,n_1}, n_{2,n_1})$
1	2	4	(2,4)-(2,4)-(2,4)-(2,4)
2	3	4	(3,4)-(3,4)-(3,4)-(3,4)
3	4	4	(4,4)-(4,4)-(4,4)-(4,4)
4	2	5	(2,5)-(2,5)-(2,5)-(2,5)-(2,5)
5	3	5	(3,5)-(3,5)-(3,5)-(3,5)-(3,5)
6	4	5	(4,5)-(4,5)-(4,5)-(4,5)-(4,5)
7	5	5	(5,5)-(5,5)-(5,5)-(5,5)-(5,5)
8	2	6	(2,6)-(2,6)-(2,6)-(2,6)-(2,6)-(2,6)
9	3	6	(3,6)-(3,6)-(3,6)-(3,6)-(3,6)-(3,6)
10	4	6	(4,6)-(4,6)-(4,6)-(4,6)-(4,6)-(4,6)
11	5	6	(5,6)-(5,6)-(5,6)-(5,6)-(5,6)-(5,6)
12	6	6	(6,6)-(6,6)-(6,6)-(6,6)-(6,6)-(6,6)
13	2	7	(2,7)-(2,7)-(2,7)-(2,7)-(2,7)-(2,7)-(2,7)
14	3	7	(3,7)-(3,7)-(3,7)-(3,7)-(3,7)-(3,7)-(3,7)
15	4	7	(4,7)-(4,7)-(4,7)-(4,7)-(4,7)-(4,7)-(4,7)
16	5	7	(5,7)-(5,7)-(5,7)-(5,7)-(5,7)-(5,7)-(5,7)
17	6	7	(6,7)-(6,7)-(6,7)-(6,7)-(6,7)-(6,7)-(6,7)
18	7	7	(7,7)-(7,7)-(7,7)-(7,7)-(7,7)-(7,7)-(7,7)
19	2	8	(2,8)-(2,8)-(2,8)-(2,8)-(2,8)-(2,8)-(2,8)-(2,8)
20	3	8	(3,8)-(3,8)-(3,8)-(3,8)-(3,8)-(3,8)-(3,8)-(3,8)
21	4	8	(4,8)-(4,8)-(4,8)-(4,8)-(4,8)-(4,8)-(4,8)-(4,8)
22	5	8	(5,8)-(5,8)-(5,8)-(5,8)-(5,8)-(5,8)-(5,8)-(5,8)
23	6	8	(6,8)-(6,8)-(6,8)-(6,8)-(6,8)-(6,8)-(6,8)-(6,8)
24	7	8	(7,8)-(7,8)-(7,8)-(7,8)-(7,8)-(7,8)-(7,8)-(7,8)
25	8	8	(8,8)-(8,8)-(8,8)-(8,8)-(8,8)-(8,8)-(8,8)-(8,8)

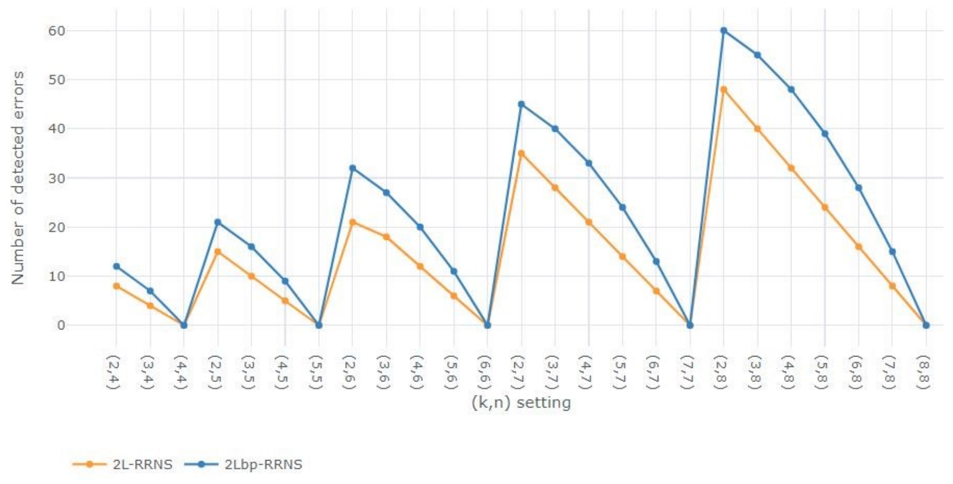


Figure 17. Error detection in 2L-RRNS and 2Lbp-RRNS

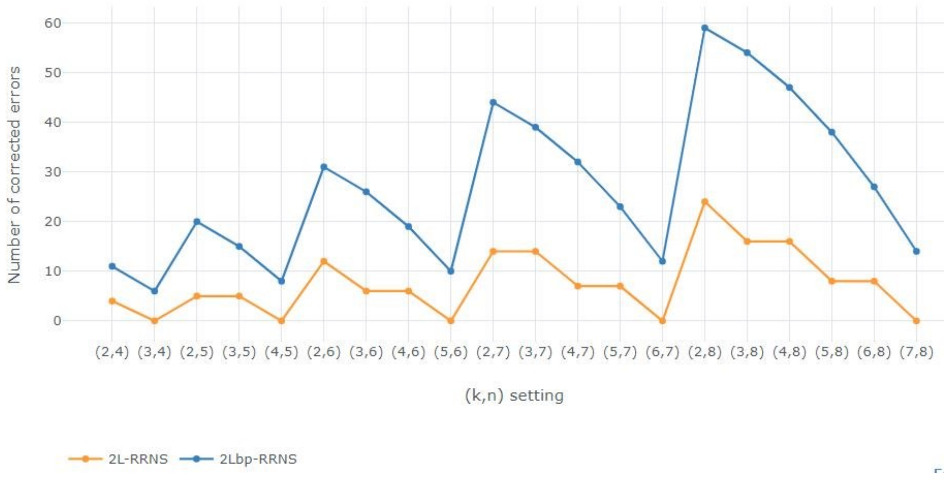


Figure 18. Error correction in 2L-RRNS and 2Lbp-RRNS

3.8 Implementation of MRC-RRNS algorithm for a 2L-RRNS

MRC-RRNS algorithms are based on the Mixed-Radix Systems (MRS) - a non-standard positional numeral system in which the numerical base varies from position to position (Huang, 1983). The basic implementations for forwarding conversion are based on a neural-like network architecture named Finite Ring Neuronal Network (FRNN) presented in Zhang et al., (1990).

Figure 19 shows the general interpretation of this architecture: a parallel, interconnected network of simple elements. It consists of two important components: 1) the neuronal processing elements capable of performing basic operations and 2) weights, representing the knowledge of the system. All finite ring arithmetic like addition, multiplication, and their combination can be reduced to this architecture.

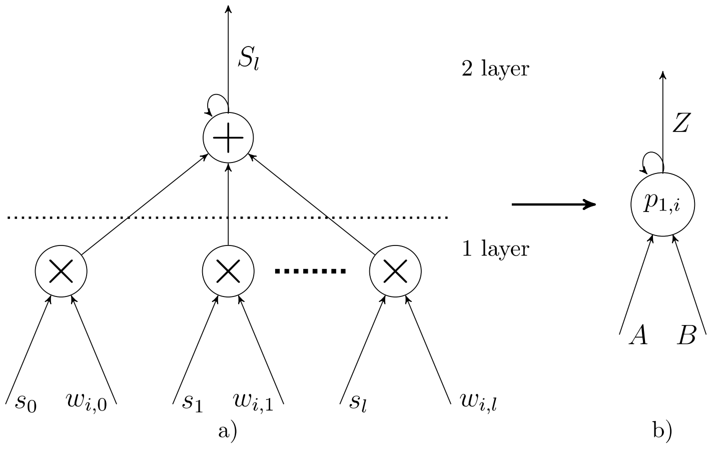


Figure 19. FRNN architecture (a) and its symbolic mapping (b)

The simple FRNN architecture is based on the Pascal method of finding the division remainder and the window method. Original data S is represented as $S = s_l | s_{l-1} | .. | s_0$, where "|" is L-bits string concatenation of s_i. L defines the window size $L \in \{8,16\}$. $w_{i,j} = \left|2^{L \cdot j}\right|_{p_{1,i}}$ are synaptic weights.

FRNN consists of two layers: the first one is the prefabricated layer on which the product s_i is calculated by the synaptic weight $w_{i,j}$. On the second computational layer, the sum of the values and remainder of the division is calculated by modulo $p_{1,i}$. Thus, FRNN can be described using the following formula (25):

$$S_i = \left|\sum_{j=0}^{l} s_j \cdot w_{i,j}\right|_{p_{1,i}} \tag{25}$$

The literature widely uses various algorithms to recover the original data: CRT (Szabo & Tanaka, 1967), Wang method (Yuke Wang, 2000), MRS (Huang, 1983), and Approximate method (N. I. Chervyakov et al., 2014).

CRT uses the computationally complex operation of finding the remainder of the division by the RRNS range to convert numbers. On the one hand, an approximate method reduces its complexity. The approximate method is based on replacing absolute values by relative values and replacing the operation of the division with the remainder of the general form by trial division. However, to obtain the correct value in the approximate method, it is necessary to increase the size of the coefficients from $\lceil \log_2 P \rceil$ to $\lceil \log_2 (P \cdot \rho) \rceil$, where $\rho = -n_1 + \sum_{i}^{n_1} p_{1,i}$, which eliminates the resulting gain. On the other hand, the recursive doubling Wang method also reduces CRT computational complexity. The Wang method reduces the size of the divisor from P to $\sqrt{P}$. But at the same time, the number of reminders of the division is increased from one to $\lceil \log_2 n_1 \rceil$.

Alternative solutions for converting from RRNS to binary are MRS-based algorithms. The decoding phase using the MRC-RRNS algorithm consists of two-stages. The number is converted from RRNS to MRS in the first stage, then from MRS to binary.

To reduce the computational complexity of the first stage, we propose a modification of the transfer algorithm from RRNS to BNS based on CRT and the FRNN. The second stage of the transition from MRS to binary is implemented using the Convolutional Neural Network (CNN).

Figure 20, shows a Decoding Neural Network (DNN) architecture for conversion from 1L-RRNS to binary consisting of two layers: FRNN and CNN.

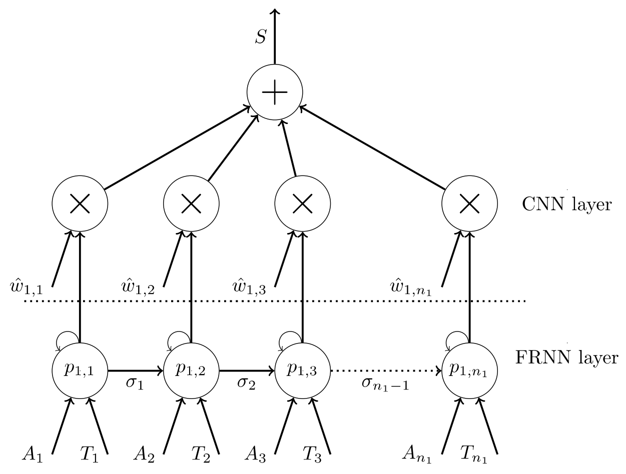


Figure 20. DNN architecture for decoding from 1L-RRNS to binary

The RRNS residues can be converted to MRS and then to binary $S \xleftarrow{MRS} \hat{S} = [\hat{s}_1, \dots, \hat{s}_{n_1}]$ by:

$$\mathrm{S} = \sum_{i=1}^{n_1} \hat{s}_i \widehat{\mathrm{w}}_{1,i} \tag{26}$$

where $\widehat{\mathrm{w}}_{1,i}$ are the radices and $\hat{s}_{1,i}$ are the MRS digits, where $0 \le \hat{s}_{1,i} < p_{1,i}$ and $\widehat{\mathrm{w}}_{1,i} = \prod_{j=1}^{i-1} p_{1,j}$.

To recover S, the classic MRC is formulated as follows:

$$S = \hat{s}_1 + \hat{s}_2 p_{1,1} + \hat{s}_3 p_{1,1} p_{1,2} + \cdots + \hat{s}_n p_{1,1} p_{1,2} \dots p_{1,n_1-1}$$

The MRC digits can be computed as:

$\hat{s}_1 = S_1$

$\hat{s}_2 = \left| (S_2 - \hat{s}_1) \left| p_{1,1}^{-1} \right|_{p_{1,2}} \right|_{p_{1,2}}$

$$\hat{s}_3 = \left|\left((S_3 - \hat{s}_1)\left|p_{1,1}^{-1}\right|_{p_{1,3}} - \hat{s}_2\right)\left|p_{1,2}^{-1}\right|_{p_{1,3}}\right|_{p_{1,3}}$$

$$\hat{s}_n = \left|\left(..(S_n - \hat{s}_1)\left|p_{1,1}^{-1}\right|_{p_{1,n_1}} - \hat{s}_2\right)\left|p_{1,2}^{-1}\right|_{p_{1,n}} - \cdots - \hat{s}_{1,n_1-1})\left|p_{1,n_1-1}^{-1}\right|_{p_{1,n_1}}\right|_{p_{1,n_1}}$$

Then, a positive number in the interval $[0, P-1]$ can be uniquely represented.

MRS reduces the computational complexity of 1L-RRNS to BNS conversion by eliminating the operation of finding the remainder of division by P, by calculating $\hat{s}_i$. The computational complexity of the $\hat{s}_i$ is quadratic of n_1. The simultaneous use of MRS and CRT ideas to translate from 1L-RRNS to the BNS allows speeding up the algorithm.

Let $B_i = \left|P_i^{-1}\right|_{p_{1,i}} \cdot P_i$ is orthogonal basis 1L-RRNS, where for all $i = \overline{1, n_1}$: $P_i = P/p_{1,i}$. For any $i = \overline{1, n_1}$: B_i is represented in the MRS as $B_i \overset{MRS}{\longleftarrow} \widehat{B}_i = [\widehat{b}_{i,1}, \dots, \widehat{b}_{i,n_1}]$, $T_i = (\widehat{b}_{1,i}, \dots, \widehat{b}_{i,i})$ is a tuple of coefficients in MRS representation, $A_i = (\underbrace{S_i, S_i, \dots, S_i}_{i\ times})$ and $\sigma_i = \left\lfloor \frac{1}{p_{1,i}} \cdot \sum_{j=1}^{i} S_j \cdot b_{j,i} \right\rfloor$ is the bias.

Chapter 4. Experimental analysis and results

This chapter introduces the results from the experimental evaluation of the Mignotte, Asmuth-Bloom, AR-RRNS, and WA-RNSS described in Chapter 2, and a two level model (2L-RRNS and 2Lbp-RRNS) described in Chapter 3. We assess the schemes' performance based on their encoding and decoding speeds, storing and extraction speeds, and redundancy.

4.1 Experimental setup

To evaluate the algorithms, we consider a data storage system in a multi-cloud environment as our simulation scenario. To analyze the behavior of the system, we use zip files compose of images, PDF files, text files, and photos, with a size of 100MB. We consider the characteristics of the Cloud storage providers of those presented in Section 3.2.

We perform the experiments on the server Express x3650 M4, with two Xeon IvyBridge processors E5-2650v2, a default clock speed of 2.6GHz. Each processor has eight Cores and two threads per core (16 with hyperthreading), level 1 memory of 32kB, level 2 of 256kB, level 3 of 20MB. Two NUMA domains of 32 GB each, with a total memory of 64GB.

The server operating system is CentOS Linux release 7.1.1503. We developed the system using the Java (JDK 12.0.1) programming language.

4.2 Analysis of 1L-RRNS

4.2.1 Evaluation of input sizes variation

To understand if the size of the input file influences the speed-related criteria, we run several experiments using four files with sizes of 10MB, 50MB, 100MB, and 200MB. We study a 1L-RRNS architecture consisting of eight Cloud storage and the schemes Mignotte, AR-RRNS, and Asmuth-Bloom. We consider several scenarios according to different access structures.

Figure 21 and Figure 22 summarize the time required for encoding. We observe that the overall behavior of the system is the same for the encoding process independently of the input size. Mignotte and AR-RRNS follow the same encoding process, while Asmuth-Bloom have the extra step where it adds "noise" into the input data. If we focus on a fixed number of clouds, for example, $n = 6$, we can see that the encoding time decreases as k increases. These results are due to the schemes creates more share but of smaller size. Mignotte and AR-RRNS reach 72.80 sec for a 200MB file and only 3.63 sec for a 10MB file for the access structure (2,6).

Conversely, Asmuth-Bloom reaches its highest values at the access structure (6,6) with 191.03 sec for 200MB and 9.435 sec for 10MB. Contrary to the other two schemes, the shares in Asmuth-Bloom do not decrease as k increases, since they do not depend on k.

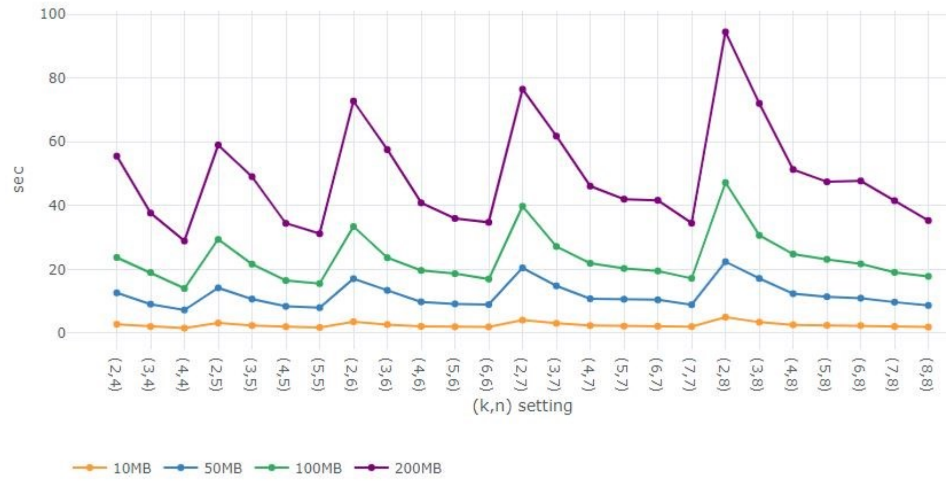


Figure 21. Encoding speeds of Mignotte and AR-RRNS for files size 10MB, 50MB, 100MB, and 200MB

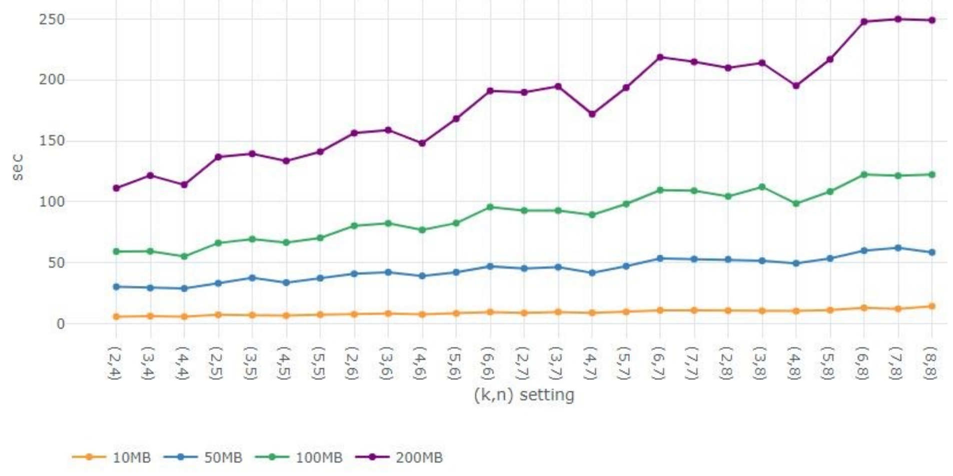


Figure 22. Encoding speeds of Asmuth-Bloom for files size 10MB, 50MB, 100MB, and 200MB

In the decoding phase, we can observe a significant difference in the behavior of AR-RRNS and Mignotte; see Figure 23 and Figure 24. AR-RRNS presents a more balanced performance with an average time of 2.45 sec for 10MB file and 48.88 sec for the 200MB file. On the other hand, Mignotte and Asmuth-Bloom use classic CRT to recover the data; in Figure 25, we show that as we move into a bigger value of k, the decoding time should increase. For an access structure (6,6), Mignotte reaches a maximum value of 61.317 sec for a 200MB file and 3.095 sec for the 10MB file, while Asmuth-Bloom 316.174 sec for a 200MB file and 15.311 sec for a file of 10MB.

Finally, based on the experiment results, the input file size does not affect the schemes' behavior. Besides, we calculate at which factor the time increases depending on the size of a file. For example, we calculate the growth factor taking the results of the 100MB file and the results of 200MB; that is, the encoding and decoding times for a 200MB file are 2x slower regarding a 100MB file. For this reason, for the rest of the experiments, we use a 100MB file to test the system since we can extrapolate its results for other input sizes.

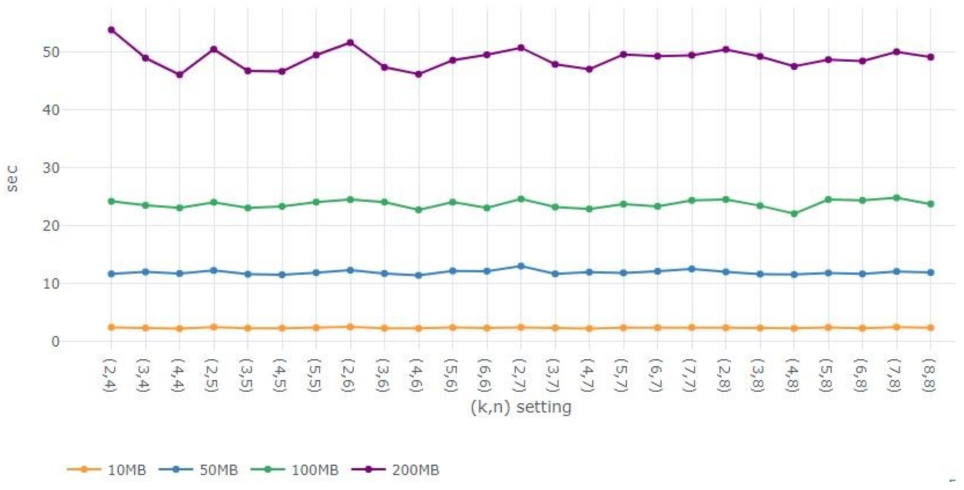


Figure 23. Decoding speeds of AR-RRNS for files size 10MB, 50MB, 100MB, and 200MB

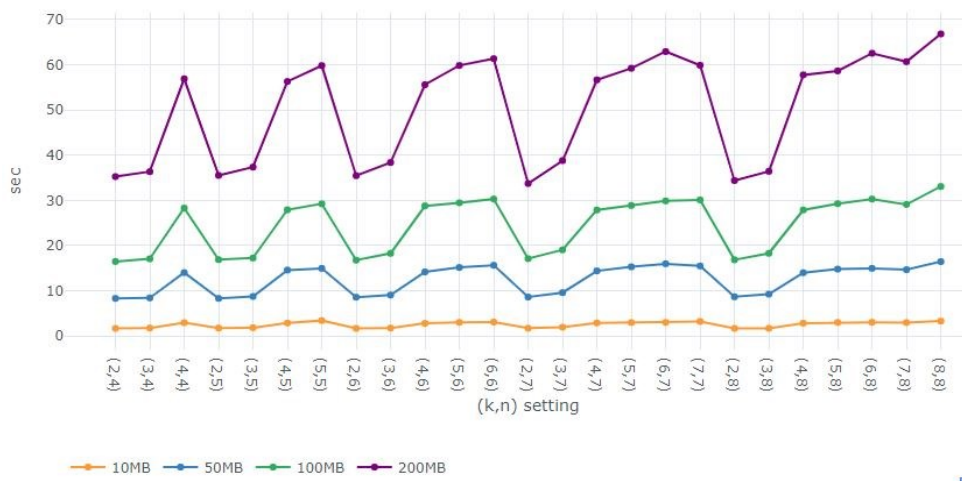


Figure 24. Decoding speeds of Mignotte for files size 10MB, 50MB, 100MB, and 200MB

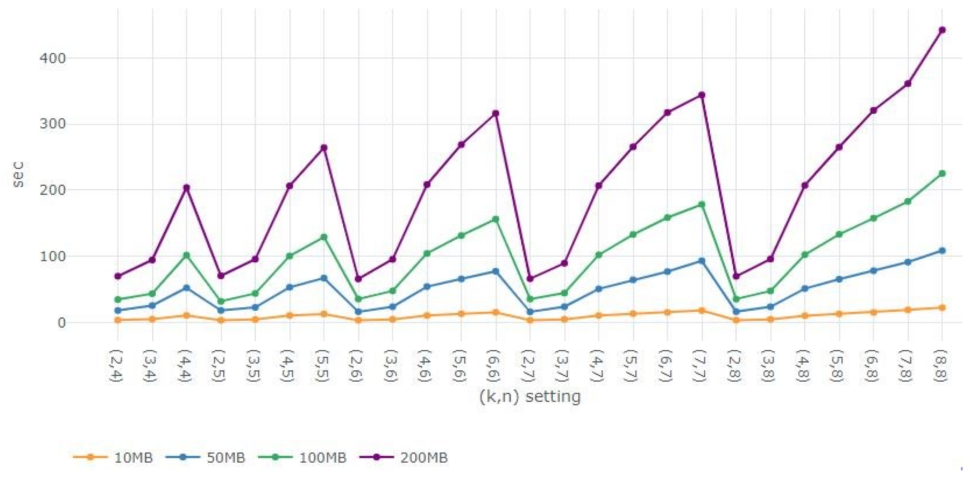


Figure 25. Decoding speeds of Asmuth-Bloom for files size 10MB, 50MB, and 100MB

4.2.2 Speed evaluation of Asmuth-Bloom, Mignotte, and AR-RRNS

We demonstrate the performance of the system in terms of speeds. Figure 26 depicts the encoding rates for Asmuth-Bloom, Mignotte, and AR-RRNS. As we establish before, Mignotte and AR-RRNS have the same encoding phase; both schemes read the input file with a calculated byte-block size from the dynamic range based on k, as described in the section 3.5.1.1. While Asmuth-Bloom calculates the byte-block size based on the value of p_0.

From Figure 26, we can observe that for access structure (2,7), Mignotte and AR-RRNS have a rate of 2.5 MB/s and Asmuth-Bloom of 1.08 MB/s. Moreover, for (7,7), Mignotte and AR-RRNS have a speed of

5.8 MB/s and 0.917 MB/s for Asmuth-Bloom. Asmuth-Bloom has more computational overhead because it adds the "noise" factor to increase the system's security causing slower encoding. Mignotte and AR-RRNS are, on average, 4x faster than Asmuth-Bloom.

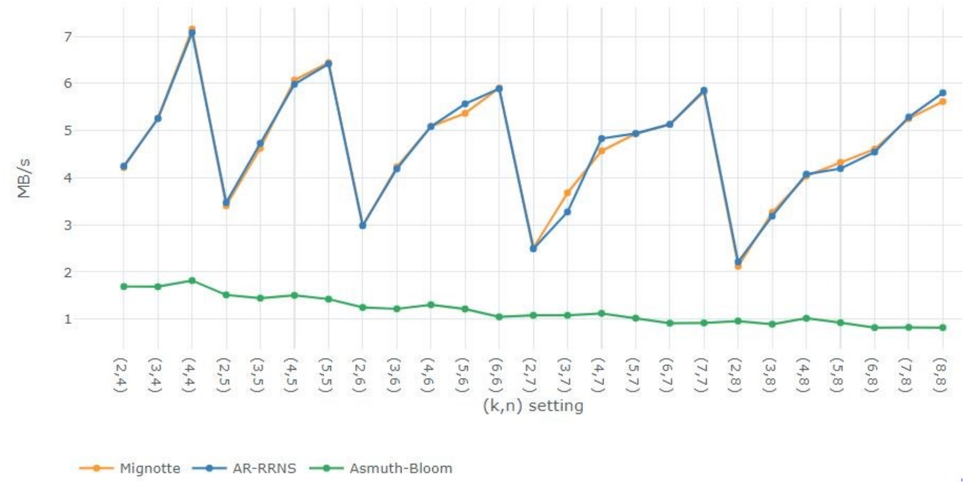


Figure 26. Encoding speeds of Asmuth-Bloom, Mignotte, and AR-RRNS

We already see that when k reaches n, the share sizes decrease for Mignotte and AR-RRNS schemes requiring less encoding time. For example, in (2,8)-Mignotte, the share's size is 104857606 MB, while for an (8,8)-Mignotte is 26214406 MB. For every k we have the same share size. Therefore, in Table 7, we only listed the access structure when $n = 8$. In contrast, the shares of Asmuth-Bloom do not change with k, since they depend solely on the value p_0.

As described in Section 2.3.3, RRNS's reliability capability depends on the redundancy of the system. So, according to different redundancy degrees r introduces overhead in terms of file size. We can use the redundancy as a metric to assess a scheme's practicability, especially in real Cloud environments, since we may be dealing with storage fees, transmissions delays, etc.

Table 7. Size of shares of Mignotte, AR-RRNS, and Asmuth-Bloom considering an input file of 104857600 bytes (Bytes)

Access structure	Sizes of shares	
	Mignotte	Asmuth-Bloom
(2,8)	104857606	209715206
(3,8)	69905072	209715206
(4,8)	52428806	209715206
(5,8)	41943046	209715206
(6,8)	34952538	209715206
(7,8)	29959320	209715206
(8,8)	26214406	209715206

Figure 27 depicts the redundancy of the system. Mignotte and AR-RRNS have the same redundancy; their higher values are when $k = 2$, since the share sizes are almost the same size as the original data. Meanwhile, when $k = n$ the output file sizes are the smallest with a minimum redundancy of two. For Asmuth-Bloom, the shares are of the same size for all the access structure. Hence, for access structures with $n = 8$, the redundancy of Asmuth-Bloom is 16, which is 8x bigger than Mignotte and AR-RRNS.

The redundancy of Asmuth-Bloom has a significant impact if we consider memory space (storage space payed to a Cloud provider); as we can see in Figure 36, it is a substantial difference between the final output size versus the original input size.

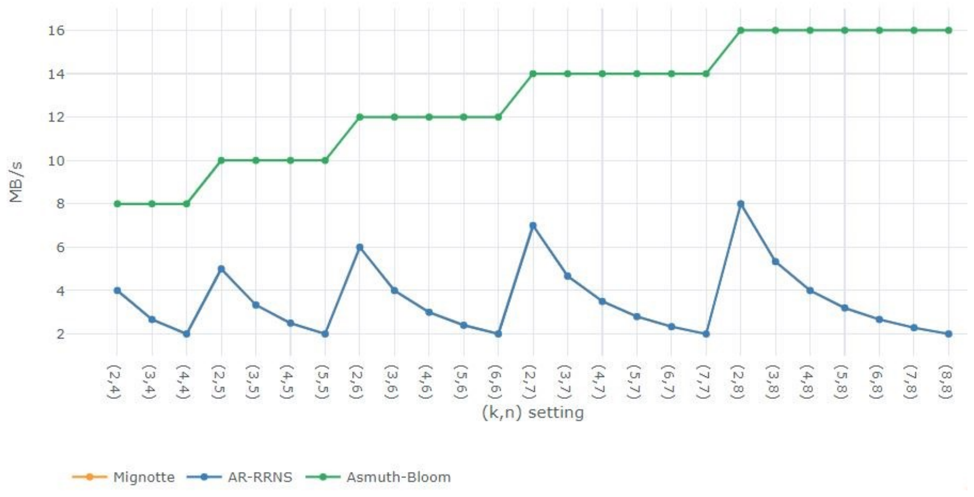


Figure 27. Redundancy of Asmuth-Bloom, Mignotte, and AR-RRNS

The three evaluated schemes are CRT-based; their decoding performance is shown in Figure 28. We can see that AR-RRNS has a smaller and more balanced range of speeds than Mignotte. Its values go from 4.02MB/s to 4.52MB/s, while Mignotte's ranges are 3.02MB/s to 6.22MB/s. So, AR-RRNS may not reach values higher than Mignotte; but its performance is more stable.

For Asmuth-Bloom, the "noise" factor also impacts the decoding phase, reaching lower values than the two other schemes. The highest values for Asmuth-Bloom are in access structures where $k = 2$. For decoding, Mignotte outperformed on average 3x Asmuth-Bloom, and AR-RRNS tops it 4.2x.

4.2.2.1 Comparison with algorithm MRC-RRNS

As mentioned in Section 2.3, there are two main methods to recover the original data from its residual representation: CRT and MRC. In the previous sections, we analyze the behavior of three SSS based on CRT: Asmuth-Bloom, Mignotte, and AR-RRNS. In this section, we analyze the algorithm MRC-RRNS; we give its detailed description in Section 3.8. MRC-RRNS is an algorithm based on MRS using MRC as a recovery method. We evaluate two versions named MRC8-RRNS, which uses an 8-bit word, and MRC16-RRNS, which uses a 16-bit word.

In Table 8, we present the encoding and decoding speeds of MRC[8,16]-RRNS. We can see that MRC16-RRNS reaches speeds up to 43 MB/s for encoding and 36 MB/s for decoding. The version of the MRC is optimize based on the notion of FRNN and CNN, and we observe that this implementation improves the encoding and decoding speeds compared to classic CRT-schemes.

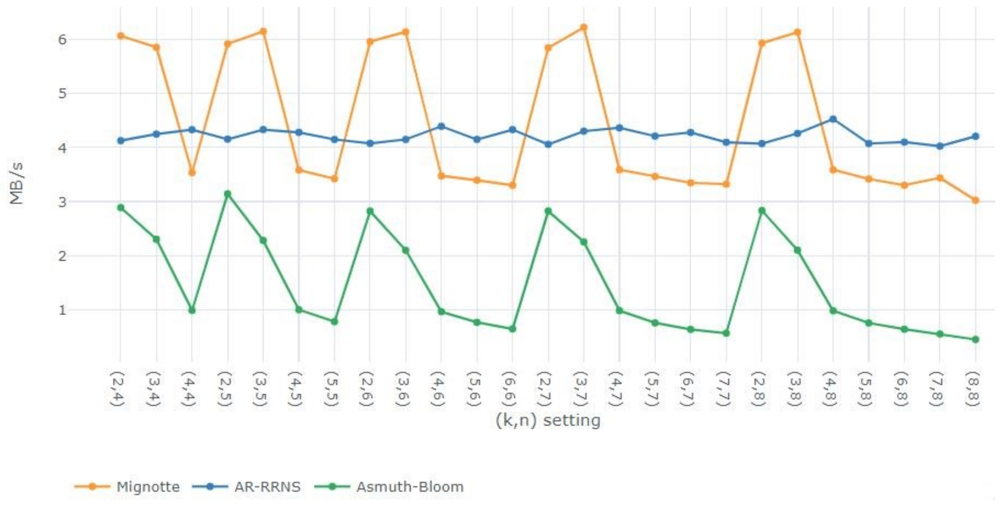


Figure 28. Decoding speeds of Asmuth-Bloom, Mignotte, and AR-RRNS

To highlight that MRC-RRNS outperforms the other schemes, we calculate the average performance degradation with the proposal of Tsafrir et al., (2007), see Section 3.4. We calculate the average of all the measurements of each algorithm and the best-found value for each access structure. Finally, we determine which algorithm has the best performance in terms of storing and extraction speeds.

Table 8. Encoding and decoding speeds of MRC[8,16]-RRNS

Access structure	Encoding		Decoding	
	MRC8	MRC16	MRC8	MRC16
(2,4)	18.68461	21.63099	24.27775	32.42544
(3,4)	28.29656	32.89475	21.94909	33.88683
(4,4)	38.36515	41.70144	21.41329	33.7838
(2,5)	15.80029	18.16531	24.60631	33.70409
(3,5)	23.29374	33.12357	21.85793	34.10643
(4,5)	33.69274	36.58984	20.78571	34.14136
(5,5)	41.78857	47.80117	17.8859	34.28181
(2,6)	13.34223	14.11234	24.36055	33.84096
(3,6)	22.52761	23.58492	22.48202	30.89282
(4,6)	27.2777	29.03602	19.89259	34.12971
(5,6)	31.35781	35.24852	17.16739	31.64558
(6,6)	38.16922	40.40406	17.17624	30.65605
(2,7)	11.39991	15.01277	27.1887	36.12718
(3,7)	19.14243	20.91614	22.49214	35.21128
(4,7)	25.60413	26.09605	20.93803	35.16176
(5,7)	28.17696	34.59012	18.01478	33.85242
(6,7)	35.00177	37.57987	17.80628	31.22074
(7,7)	33.85242	43.0849	17.58088	29.47246
(2,8)	9.665576	10.51525	26.41311	36.1011
(3,8)	14.18843	17.70226	22.40144	33.48963
(4,8)	21.34017	22.86238	19.84916	34.50657
(5,8)	26.50412	28.94357	16.9119	32.89475
(6,8)	32.15436	32.52034	17.24436	31.51593
(7,8)	32.0513	37.29953	16.25224	29.39449
(8,8)	40.48048	42.12302	15.16991	26.92516

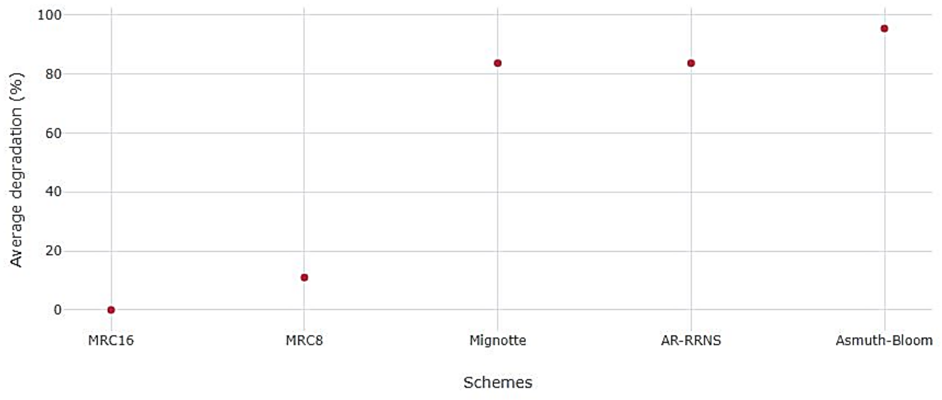


Figure 29. Average degradation of encoding speeds of each scheme using a 100MB input file

As expected, in Figure 29 and Figure 30, we observe that the best algorithm is MRC16-RRNS, with the best coding and decoding performance. On the other hand, Mignotte and AR-RRNS are the best algorithms based on CRT.

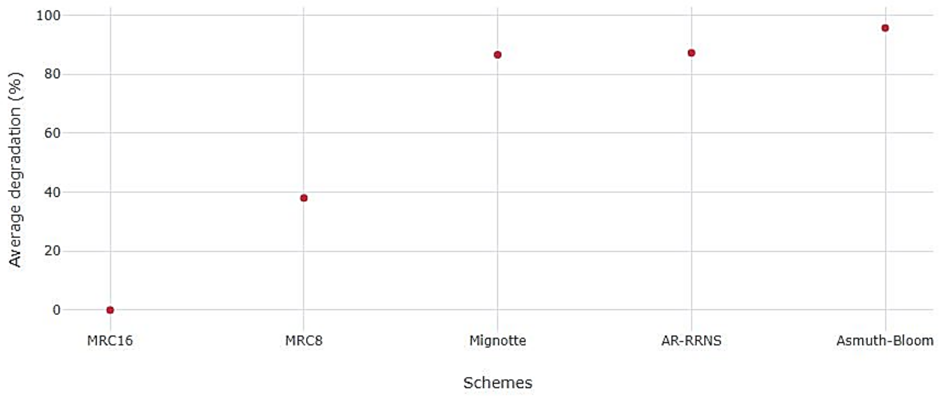


Figure 30. Average degradation of decoding speeds of each scheme using a 100MB input file

4.2.2.2 Uploading and downloading performance

Here, we analyze the performance of the schemes in terms of storing/extraction speeds in a multi-cloud scenario. In Table 4 of Section 3.2, we show measured upload and download speed (MB/s) of cloud providers.

We calculate the maximum and minimum values that can be achieved based on the worst and best upload and download access speeds. For these cases, we assume that all clouds have the same speed, either best or worst. It can give us the range of speeds that we can expect so that any behavior will be within these values.

When the uploading time is taking into account, we calculate the Storing speed (Vs) using equation (16), $V_s = \frac{size(D)}{T_s}$. In this case we assigned to all the clouds the best access speed of 3.45 MB/s and the worst access speed of 0.11 MB/s. For the Extraction speed, we use the equation V_{ex} using equation (17), $V_{ex} = \frac{sise(D)}{T_{ex}}$, here the best access speed was 3.32 MB/s and the worst of 0.68 MB/s. Table 9 and Table 10, show the worst and best times for uploading and downloading from the cloud providers, for Mignotte, Asmuth-Bloom, and MRC16-RRNS. For example, for Mignotte the download times are in 0.2941 sec or the worst case and 0.602 sec for the best case, while the decoding times are in the 16.056 sec and 33.062 sec range.

The upload and download times are conditioned by the access speeds and the size of the shares depending each of the schemes. Besides, to calculate the downloading time, we assume that only k shares are used. From Figure 31 to Figure 33, we can observe that the extraction speeds (Vex) demonstrate similar values. This is because the decoding time has a greater impact than the access times.

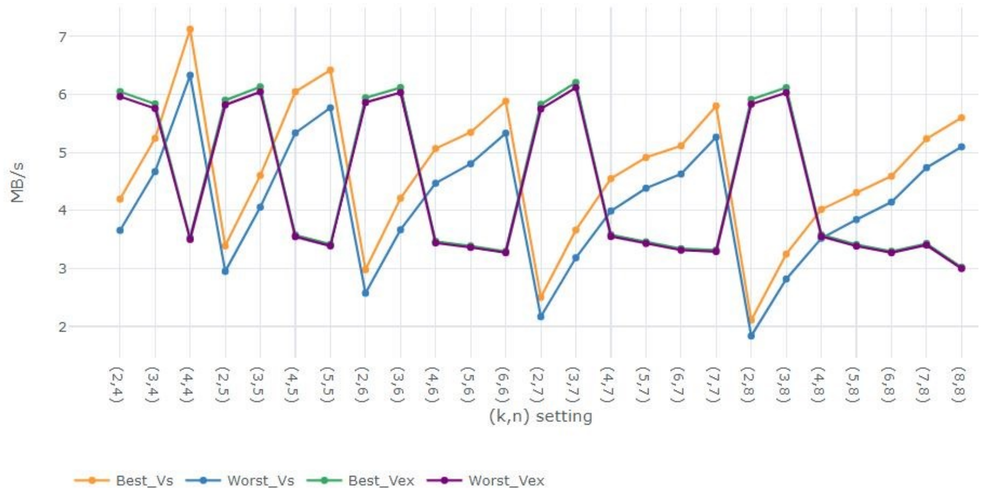


Figure 31. Best and Worst storing and extraction speeds for Mignotte

Table 9. Best and worst uploading times for Mignotte, Asmuth-Bloom and MRC16-RRNS

Access structure	Best uploading time (sec)			Worst uploading time (sec)		
	Mignotte	Asmuth-Bloom	MRC16	Mignotte	Asmuth-Bloom	MRC16
(2,4)	0.115942	0.231884	0.077295	3.636364	7.272727	2.424243
(3,4)	0.077295	0.231884	0.046377	2.424243	7.272727	1.454546
(4,4)	0.057971	0.231884	0.033126	1.818182	7.272727	1.038961
(2,5)	0.144928	0.289855	0.096618	4.545455	9.090909	3.030303
(3,5)	0.096618	0.289855	0.057971	3.030303	9.090909	1.818182
(4,5)	0.072464	0.289855	0.041408	2.272728	9.090909	1.298701
(5,5)	0.057971	0.289855	0.032206	1.818182	9.090909	1.010101
(2,6)	0.173913	0.347826	0.115942	5.454546	10.90909	3.636364
(3,6)	0.115942	0.347826	0.069565	3.636364	10.90909	2.181818
(4,6)	0.086957	0.347826	0.049689	2.727273	10.90909	1.558442
(5,6)	0.069565	0.347826	0.038647	2.181818	10.90909	1.212121
(6,6)	0.057971	0.347826	0.031621	1.818182	10.90909	0.991736
(2,7)	0.202899	0.405797	0.135266	6.363637	12.72727	4.242424
(3,7)	0.135266	0.405797	0.081159	4.242425	12.72727	2.545455
(4,7)	0.101449	0.405797	0.057971	3.181819	12.72727	1.818182
(5,7)	0.081159	0.405797	0.045089	2.545455	12.72727	1.414142
(6,7)	0.067633	0.405797	0.036891	2.121212	12.72727	1.157025
(7,7)	0.057971	0.405797	0.031215	1.818182	12.72727	0.979021
(2,8)	0.231884	0.463768	0.154589	7.272728	14.54545	4.848485
(3,8)	0.154589	0.463768	0.092754	4.848485	14.54545	2.909091
(4,8)	0.115942	0.463768	0.066253	3.636364	14.54545	2.077922
(5,8)	0.092754	0.463768	0.05153	2.909091	14.54545	1.616162
(6,8)	0.077295	0.463768	0.042161	2.424243	14.54545	1.322314
(7,8)	0.066253	0.463768	0.035674	2.077922	14.54545	1.118881
(8,8)	0.057971	0.463768	0.030918	1.818182	14.54545	0.969697

From Figure 32, we can observe that MRC16-RRNS has a gap of almost 12 MB/s for access structures where $k = \overline{n-2, n}$.

Table 10. Best and worst uploading times for Mignotte, Asmuth-Bloom and MRC16-RRNS

Access structure	Best downloading time (sec)			Worst downloading time (sec)		
	Mignotte	Asmuth-Bloom	MRC16	Mignotte	Asmuth-Bloom	MRC16
(2,4)	0.060241	0.120482	0.040161	0.294118	0.588235	0.196078
(3,4)	0.060241	0.180723	0.036145	0.294118	0.882353	0.176471
(4,4)	0.060241	0.240964	0.034423	0.294118	1.176471	0.168067
(2,5)	0.060241	0.120482	0.040161	0.294118	0.588235	0.196078
(3,5)	0.060241	0.180723	0.036145	0.294118	0.882353	0.176471
(4,5)	0.060241	0.240964	0.034423	0.294118	1.176471	0.168067
(5,5)	0.060241	0.301205	0.033467	0.294118	1.470588	0.163399
(2,6)	0.060241	0.120482	0.040161	0.294118	0.588235	0.196078
(3,6)	0.060241	0.180723	0.036145	0.294118	0.882353	0.176471
(4,6)	0.060241	0.240964	0.034423	0.294118	1.176471	0.168067
(5,6)	0.060241	0.301205	0.033467	0.294118	1.470588	0.163399
(6,6)	0.060241	0.361446	0.032859	0.294118	1.764706	0.160428
(2,7)	0.060241	0.120482	0.040161	0.294118	0.588235	0.196078
(3,7)	0.060241	0.180723	0.036145	0.294118	0.882353	0.176471
(4,7)	0.060241	0.240964	0.034423	0.294118	1.176471	0.168067
(5,7)	0.060241	0.301205	0.033467	0.294118	1.470588	0.163399
(6,7)	0.060241	0.361446	0.032859	0.294118	1.764706	0.160428
(7,7)	0.060241	0.421687	0.032437	0.294118	2.058824	0.158371
(2,8)	0.060241	0.120482	0.040161	0.294118	0.588235	0.196078
(3,8)	0.060241	0.180723	0.036145	0.294118	0.882353	0.176471
(4,8)	0.060241	0.240964	0.034423	0.294118	1.176471	0.168067
(5,8)	0.060241	0.301205	0.033467	0.294118	1.470588	0.163399
(6,8)	0.060241	0.361446	0.032859	0.294118	1.764706	0.160428
(7,8)	0.060241	0.421687	0.032437	0.294118	2.058824	0.158371
(8,8)	0.060241	0.481928	0.032129	0.294118	2.352941	0.156863

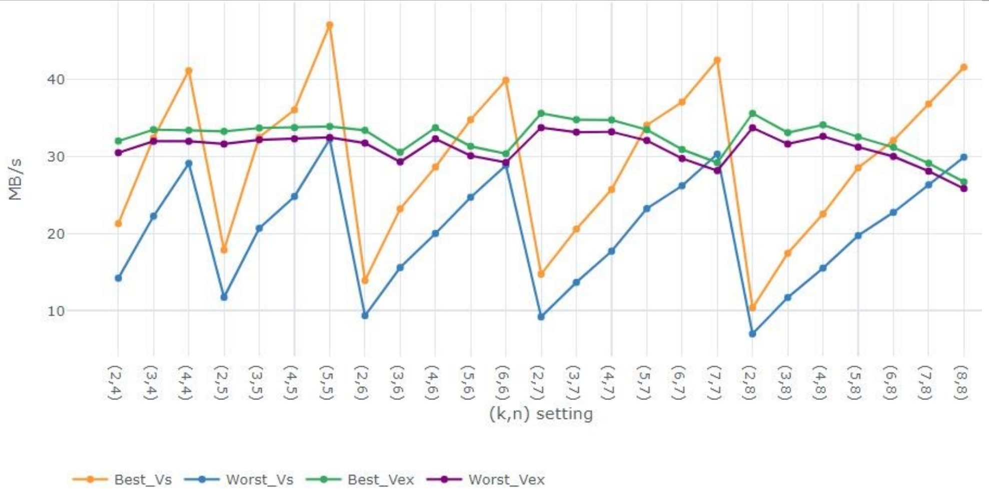


Figure 32. Best and Worst storing and extraction speeds for MRC16-RRNS

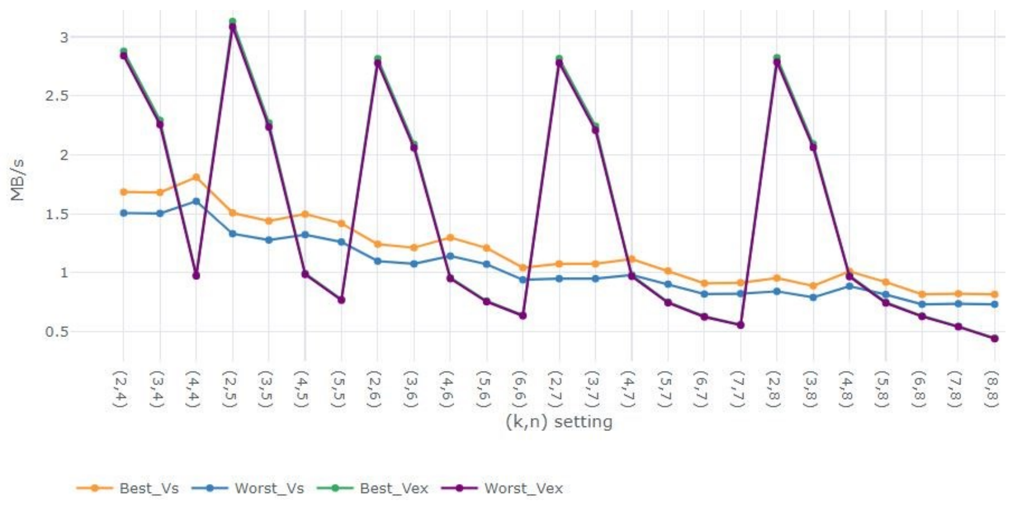


Figure 33. Best and Worst storing and extraction speeds for Asmuth-Bloom

4.2.2.3 Comparison with Reed-Solomon and AES encryption system

For comparison proposes, we implement a system named RS+AES consisting of a Reed-Solomon error-correction code (RS) and a phase of encryption using the AES algorithm to ensure the data's security (see Section 2.4.5). In this implementation, we used the public library Jerasure (Plank et al., 2007). Jerasure is a C library released in 2007 that supports the classic Reed-Solomon, but also Cauchy-RS, and Minimal Density RAID-6 coding. We took the C version and implemented a JAVA language version. For the implementation of the AES algorithm, we use the embedded Java Library on the JDK 12.0.1.

We take the input data D; we encrypt it using the AES algorithm to provide the confidentiality capability to the system since RS leaves the original data intact. Next, we use (n, m)-Reed-Solomon algorithm, where n is the number of total segments (shares in our case), m is the number of parity segments. Then, RS divides D into $n - m$ parts.

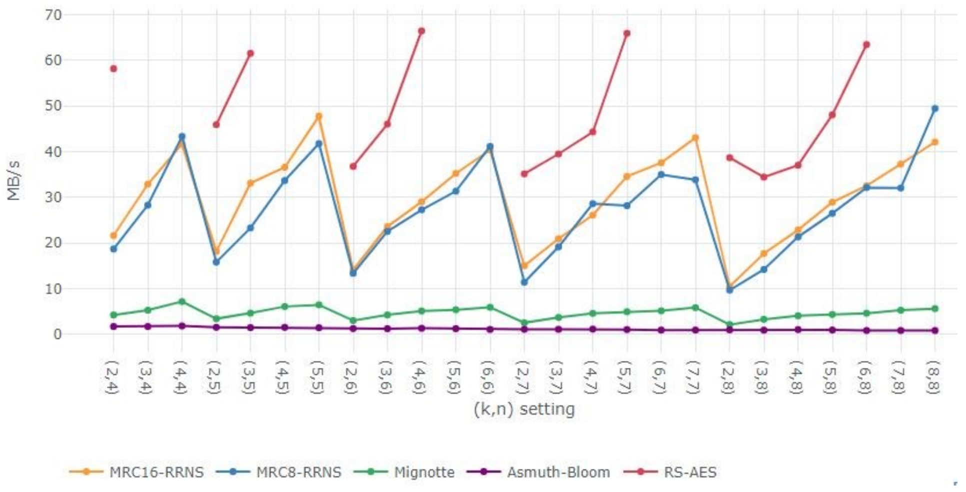


Figure 34. Encoding speeds of RS+AES vs. MRC16-RRNS, MRC8-RRNS, Mignotte and Asmuth-Bloom

We need to emphasize that RS does not have all the characteristics of a secret sharing scheme. It does not encode the input data, and it does not have the capability of performed calculations on the encrypted data as SSS CRT-based schemes do.

RS has the constraint that the minimum number of parity shares is two, $m \geq 2$. Therefore, in the experiments, we omit comparisons with access structures that violated this restriction. For example, when we are evaluating four Clouds storages, e.g. (2,4), (3,4), and (4,4), we create four shares in total. For RS's case, we need to calculate how many symbols (shares) will be data and parity.

In the case of (3,4), we create four shares, three of them are needed to restore the data, and one is a redundant share. For RS, we need to divide the data into three shares and create one parity share to maintain a total of four shares. However, for implementation purposes, we need a minimum of two parity shares for RS to works. The case of (4,4) is trivial since we do not have any parity symbol to recover the data in case of erasure or errors.

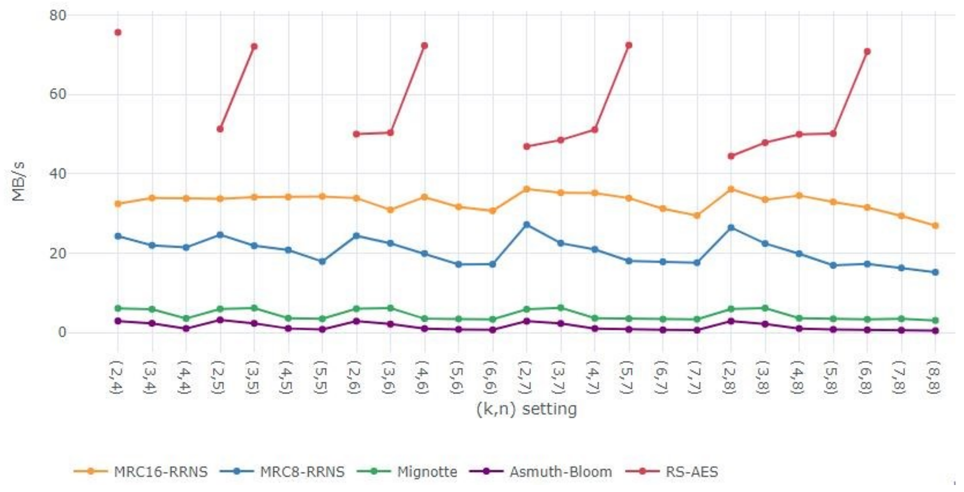


Figure 35. Decoding speeds of RS+AES vs. MRC16-RRNS, MRC8-RRNS, Mignotte and Asmuth-Bloom

In Figure 34 and Figure 35, we observe that RS+AES outperforms all the schemes. RS+AES reaches an encoding speed of 65.96 MB/s in the setting (5,7), while MRC16-RRNS is the closes scheme with a speed of 34.59 MB/s for the same configuration. For decoding, we consider that no error occurs, so RS does not have to try to recover the data; it just merely retrieve the data symbols (shares) and reconstruct the original file. This is why the decoding speed of RS+AES can reach a value of 72.41 MB/s.

In Table 11, we show the average performance degradation for all the algorithms. We observed that RS+AES is the best algorithm for encoding and decoding speeds, while Asmuth-Bloom is the worst. Even though there are access structures that are not suitable for a multi-cloud, for example, those where the number of data symbols less than 2.

Table 11. Average performance degradation (%) in terms of encoding/decoding speeds

	RS+AES	MRC16-RRNS	MRC8-RRNS	Mignotte	Asmuth-Bloom
Encoding	0	51.8	56.79	91.768	97.505
Decoding	0	38.147	59.627	90.733	96.468

The redundancy is related to the total encoded output file length and the original input length (in bytes). We refer to the total length of the encoded output file as the sum of all shares sizes for each access structure; see Figure 36. The comparison of all the schemes' redundancy shows that RS+AES has a higher redundancy for access structures with more parity symbols since the symbols are bigger. Again, we see that Asmuth-Bloom is unpractical if used with files on the MB order, see Figure 37.

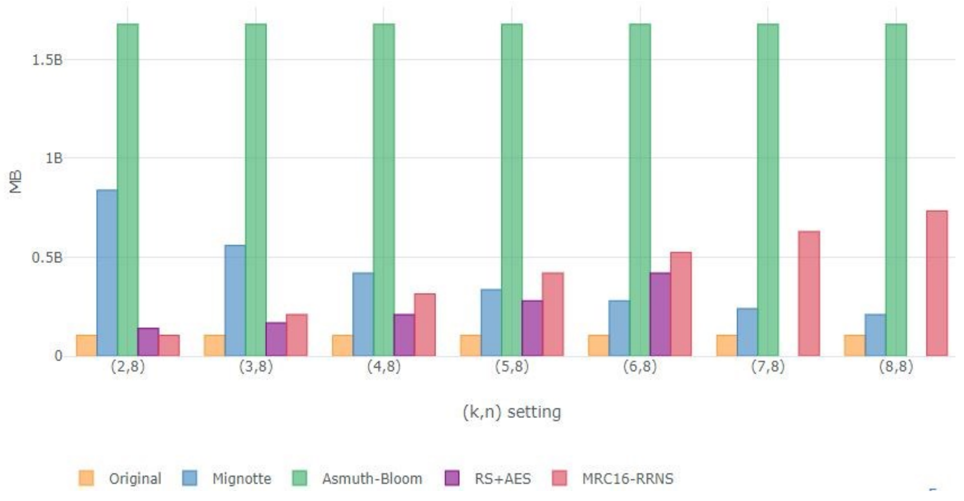


Figure 36. The length of the original file and the file encoded by Mignotte, Asmuth-Bloom, RS+AES, and MRC16-RRNS

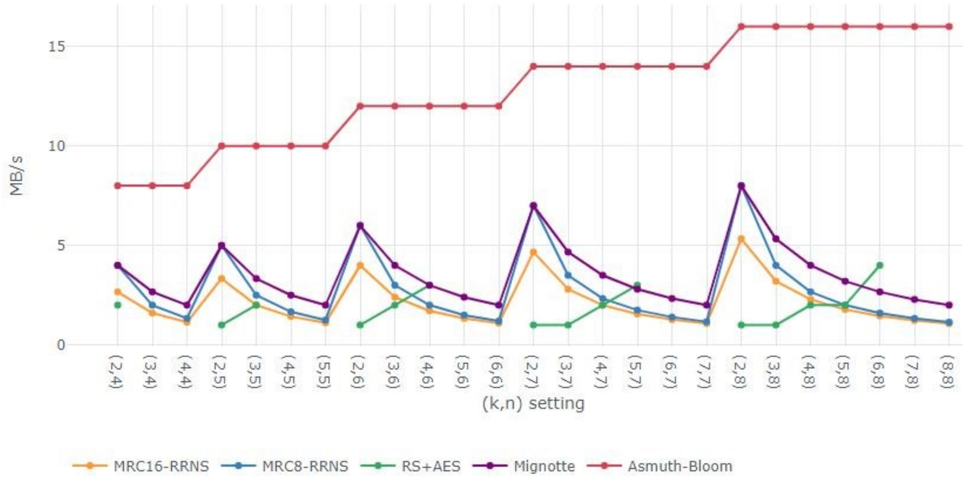


Figure 37. Redundancy of Mignotte, Asmuth-Bloom, MRC-RRNS, and RS+AES

4.2.3 Error detection-correction capability

We present the analysis of three state-of-the-art error correction codes: Modular Projection, Syndrome, and AR-ECC method based on RRNS. All of the three RRNS-based codes are explained in Section 2.4.

If we add error detection and correction code in the decoding phase, it will increase the performance of the system in terms of decoding speed, and therefore, the extraction speed. The number of combinations needed to detect which residue is erroneous and recover the original value can be significant. The error correction method could try all possible combinations. Since we can only detect a number of errors equal to the redundant moduli r, we do not consider access structures that violate this constraint.

To analyze the worst-case scenario, we assume without loss of generality that faulty share corresponds to the first modulus and that all remaining shares are correct. As mentioned before, in this work, we consider an error caused by faulty hardware, faulty software, technical malfunctions, DDoS attacks, human-made errors, providers' bankruptcy.

Table 12. Number of combinations in the worst-case scenario

Access structure	Number of combinations		
	Projection	Syndrome	AR-ECC
(2,4)	4	4	4
(2,5)	7	5	7
(2,6)	11	6	11
(2,7)	16	7	16
(2,8)	22	8	22
(3,5)	5	7	5
(3,6)	11	11	11
(3,7)	21	16	21
(3,8)	36	22	36
(4,6)	6	11	6
(4,7)	16	21	16
(4,8)	36	36	36
(5,7)	7	16	7
(5,8)	22	36	22
(6,8)	8	22	8

Table 12 shows the number of combinations performed by each CRT-based method to detect and recover the actual value. As we can see, the number of combinations grows while n increases affecting the computational overhead of the system.

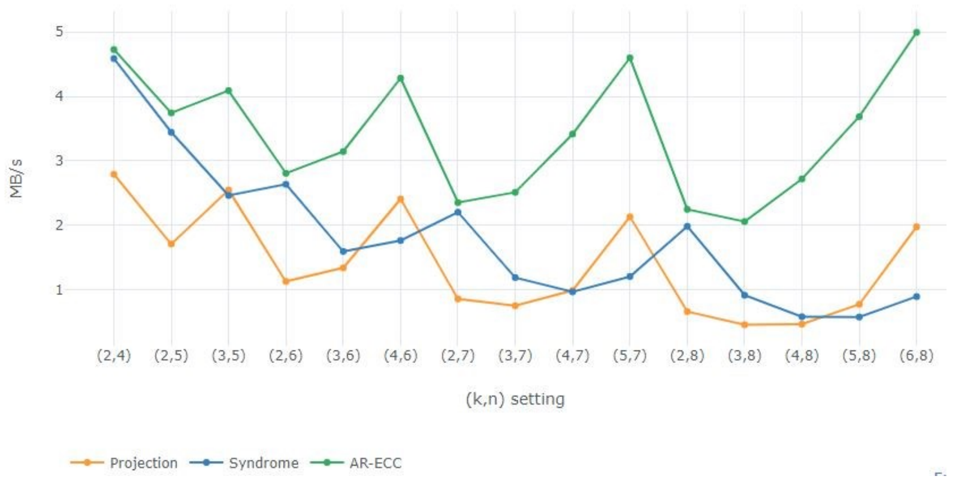


Figure 38. Decoding speeds of Mignotte with error correction in the worst-case scenario

Figure 38 and Figure 39 show the decoding speeds of Asmuth-Bloom and Mignotte in the worst case. In Figure 38, we see that for access structures (3,5), (4,6), (5,7), and (6,8), AR-ECC calculates fewer decoding combinations than Syndrome and obtains maximum values. For (6,8), Syndrome calculates more combinations than AR-ECC and Projection, resulting in a low decoding speed with a value of 0.0894MB/s.

Figure 39 shows the decoding speeds of Asmuth-Bloom. The minimum number of combinations performed by Syndrome is on settings (2,4), (2,5), (2,6), (2,7), and (2,8). Therefore, it obtains maximum values in a range from 2.92MB/s for (2,4) to 0.196MB/s for (2,8). On the other hand, AR-ECC has an average speed of 1.46MB/s for all settings.

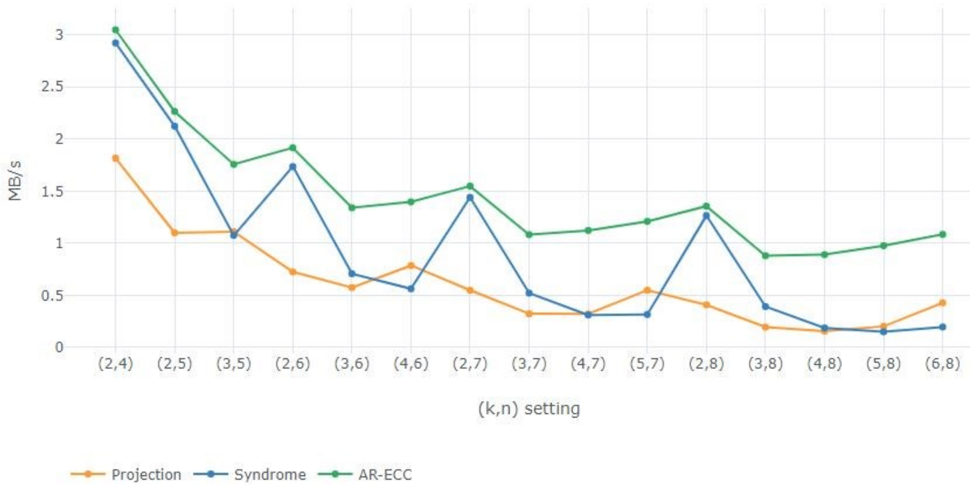


Figure 39. Decoding speeds of Asmuth-Bloom with error correction for the worst-case scenario

Overall, AR-ECC outperforms 2.33 times to Syndrome and three times to Projection when Mignotte is used. For Asmuth-Bloom, AR-ECC is 1.89 times faster than Syndrome and 2.84 times faster than Projection.

To better evaluate the performance between strategies, we calculate the performance degradation between the three error codes. Table 13 shows the performance degradation for each error correction method.

The best approach with the lowest average performance degradation has rank one. The results show that AR-ECC has the lowest degradation in all access structure settings. As established before, AR-ECC outperforms the Projection method by 68% and Syndrome by 52%, on average.

4.2.4 Performance of WA-RRNS

In this section, we evaluate the SS-WA-RRNS model (Section 3.6.2) regarding encoding/decoding speeds. First, we apply WA-RRNS into two secret sharing schemes: Mignotte and Asmuth-Bloom. Next, we apply WA-RRNS using the MRC-RRNS algorithms. Finally, we compare their behavior of all the algorithms with the results of the traditional model ES-RRNS. For the rest of the section, we will be calling the traditional model as (k, n)-System.

Table 13. Performance degradation of Mignotte and Asmuth-Bloom

Access structure	Mignotte			Asmuth-Bloom		
	Projection	Syndrome	AR-ECC	Projection	Syndrome	AR-ECC
(2,4)	44.16508	8.109143	0	44.15230	8.381715	0
(2,5)	56.99461	8.668675	0	54.97917	10.746390	0
(3,5)	53.12364	53.099270	0	52.52202	54.821010	0
(2,6)	61.45456	8.497880	0	61.64284	12.926620	0
(3,6)	67.60974	60.409690	0	67.71830	60.625530	0
(4,6)	58.99375	70.529140	0	58.93272	71.285490	0
(2,7)	67.57795	16.020970	0	66.23730	12.964840	0
(3,7)	76.33753	63.318500	0	76.91897	63.130000	0
(4,7)	78.02230	77.571460	0	77.98095	78.640890	0
(5,7)	64.76221	79.293850	0	67.84001	81.111540	0
(2,8)	70.63810	11.845980	0	73.65310	13.856010	0
(3,8)	82.53511	67.070930	0	82.70025	66.062311	0
(4,8)	86.10788	83.270700	0	86.16210	83.313020	0
(5,8)	84.01901	88.147230	0	83.99376	88.367320	0
(6,8)	71.37274	86.487450	0	70.11148	86.713600	0

Table **14** presents all the parameters for the access structures of each model. The number of Cloud storage available, the number of shares needed to recover the data, and the total number of shares encoded for the traditional (k, n)-System. It also shows the number of short shares sends to each Cloud for (n_v, k, N) WA-RRNS. This table applies to both versions of WA-RRNS.

In (n_v, k, N) -WA-RRNS, more than one short-share, is sent to each Cloud. Hence, the number of shares obtained after the encoding phase is more than in the (k, n) System with a total of 56 short-shares for an access structure of (8,8), while (k, n)-System creates only eight shares for the same equivalent access structure.

4.2.4.1 Speed evaluation of WA-RRNS with Mignotte and Asmuth-Bloom

In an access structure $(8,8)$, (k, n)-System only encodes eight shares with an encoding speed of 4.27MB/s for Mignotte and 0.6MB/s for Asmuth-Bloom. In contrast, (n_v, k, N)-WA-RRNS encodes 56 short-shares resulting in encoding speeds of 1.008MB/s for WA-Mignotte. Due to the intrinsic complexity of Asmuth-Bloom, the system crashed for access schemes when we need to create a large number of shares. As can be expected, the redundancy of (n_v, k, N)-WA-RRNS, especially of Asmuth-Bloom, is much higher than of the (k, n)-System, see Figure 41. When $N = 8$, the redundancy of Asmuth-Bloom goes from 8 to 64, which for (6,8) is an increment of 8x. Besides, the share’s sizes do not change, resulting in a very high memory cost, i.e., access structures (7,8) and (8,8).

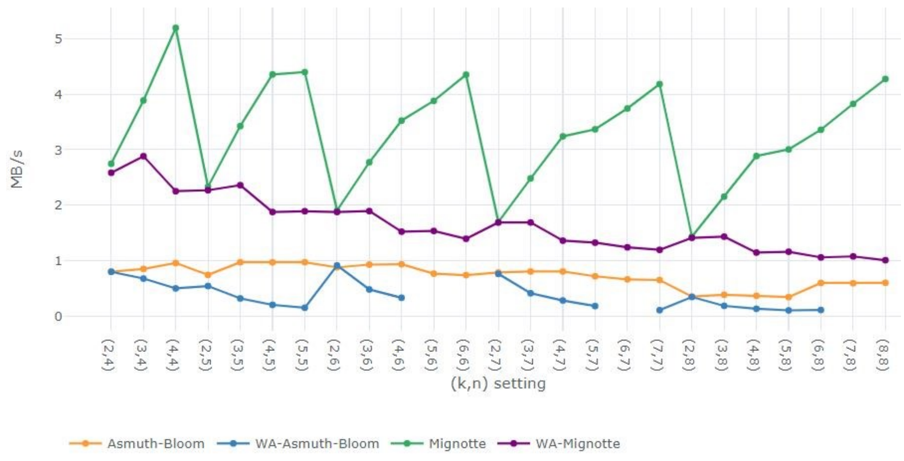


Figure 40. Encoding speeds of (n_v, k, N)-WA-RRNS and (k, n)-System

Table 14. Number of shares in traditional (k, n) system and (n_v, k, N) WA-RRNS

	$(\boldsymbol{k}, \boldsymbol{n})$ System		$(\boldsymbol{n_v}, \boldsymbol{k}, \boldsymbol{N})$ WA-RRNS		
Number of Cloud storages	*Number of shares for recovering*	*Total number of shares*	*Number of shares in each CSP*	*Number of shares for recovering*	*Total number of shares*
4	2	4	1	2	4
	3	4	2	4	8
	4	4	3	6	12
5	2	5	1	2	5
	3	5	2	4	10
	4	5	3	6	15
	5	5	4	8	20
6	2	6	1	2	6
	3	6	2	4	12
	4	6	3	6	18
	5	6	4	8	24
	6	6	5	10	30
7	2	7	1	2	7
	3	7	2	4	14
	4	7	3	6	21
	5	7	4	8	28
	6	7	5	10	35
	7	7	6	12	42
8	2	8	1	2	8
	3	8	2	4	16
	4	8	3	6	24
	5	8	4	8	32
	6	8	5	10	40
	7	8	6	12	48
	8	8	7	14	56

On the other hand, the sizes of the short-shares in (n_v, k, N)-WA-RRNS decrease while k is increased, see Table 15. We can observe that while the number of clouds increases, the encoding speeds of (n_v, k, N)-WA-RRNS decrease, reaching its lowest value when $\mathrm{N} = 8$, see Figure 40. For the overall behavior, WA-Mignotte encoding speed is, on average, almost 2x less than Mignotte, while for Asmuth-Bloom is nearly 3x slower.

Table 15. Sizes of shares of traditional system and WA-RRNS (Bytes)

Access structure $(\boldsymbol{k},\boldsymbol{n})$ system	Sizes of shares		Access structure WA-RRNS	Sizes of shares	
	Mignotte	Asmuth-Bloom		Mignotte	Asmuth-Bloom
(2,8)	104857606	209715206	(1,2,8)	104857606	209715206
(3,8)	69905072	209715206	(2,4,16)	52428806	209715206
(4,8)	52428806	209715206	(3,6,24)	34952538	209715206
(5,8)	41943046	209715206	(4,8,32)	26214406	209715206
(6,8)	34952538	209715206	(5,10,40)	20971526	209715206
(7,8)	29959320	209715206	(6,12,48)	17476272	
(8,8)	26214406	209715206	(7,14,56)	14979662	

Thus, using an SS-WA-RRNS model for schemes like Asmuth-Bloom results impractical because the amount of information that needs to be store will lead to higher costs and difficulties in uploading and downloading from the Clouds. In the case of Mignotte, the increment of the redundancy is only 6x regards (k,n)-System.

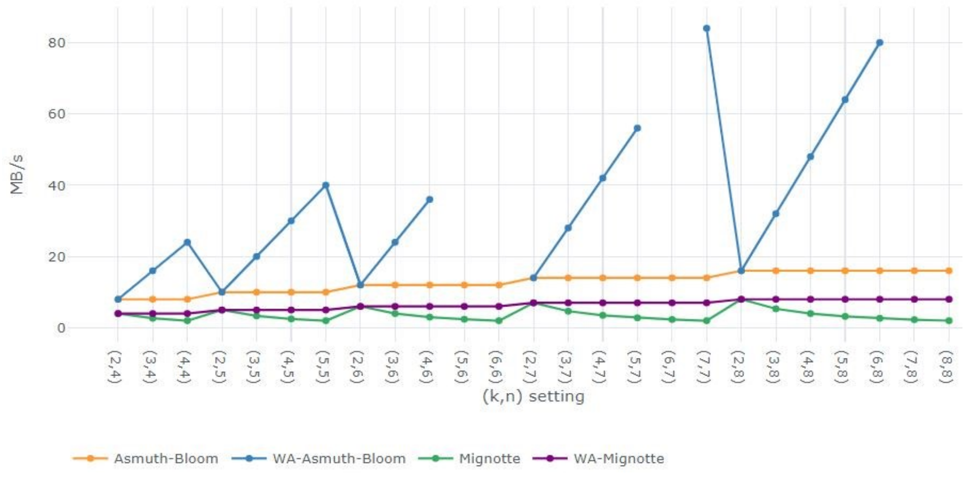


Figure 41. Redundancy of (n_v,k,N)-WA-RRNS and (k,n)-System

For the decoding phase, the (k,n)-System outperforms (n_v,k,N)-WA-RRNS almost 2x, on average, see Figure 42. Even though the shares' size is smaller for some access structures in (n_v,k,N)-WA-RRNS, it needs more short-shares to recover the data causing lower decoding speeds. In access structures where $k=2$, (n_v,k,N)-WA-RRNS uses the same number of shares and the same size.

Nonetheless, as k reaches n the decoding speeds of (n_v, k, N)-WA-RRNS for Mignotte go from 4.23MB/s to 2.65 MB/s, reducing the performance by a factor of 0.89 on average.

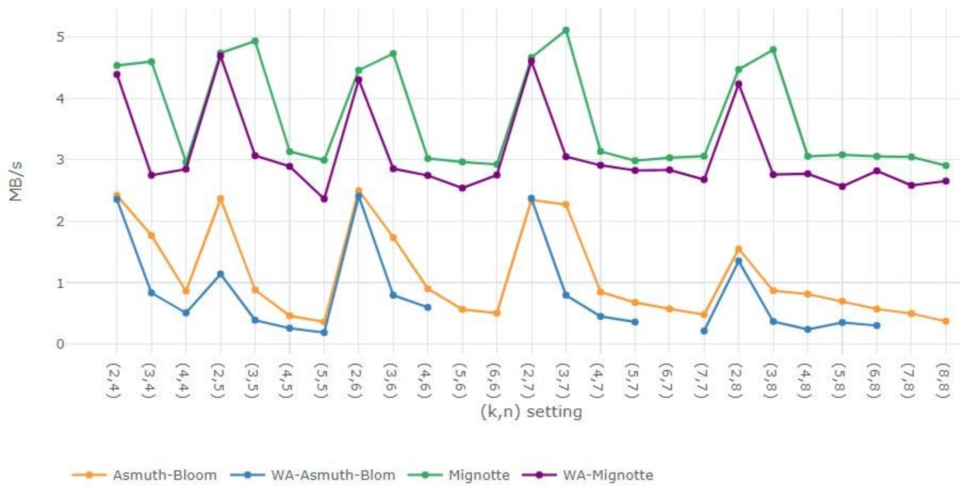


Figure 42. Decoding speeds of (n_v, k, N)-WA-RRNS and (k, n)-System

4.2.4.2 Uploading and downloading performance of WA-RRNS with Mignotte and Asmuth-Bloom

In this section, we calculate the storing and extraction speeds for WA-RRNS. Using the Cloud characteristics of Table 4 of Section 3.2, we obtain the storing speed of (n_v, k, N)-WA-RRNS and (k, n)-Systems shown in Figure 43. Comparing the encoding and storing speeds, we see that the performance of Mignotte and Asmuth-Bloom degrades almost twice as depicted in Figure 43. For (n_v, k, N)-WA-RRNS, the storing speed V_S is decreased while k and n reaches eight clouds. This is expected, since (n_v, k, N)-WA-RRNS has to send more shares. We can see that the Clouds' access speeds have a significant impact on the system's overall performance. In these experiments, we assume that under the conditions of limited Internet bandwidth, the shares are uploaded/downloaded from the clouds sequentially. On average, the decoding speeds are 2x faster than the storing speeds of both schemes.

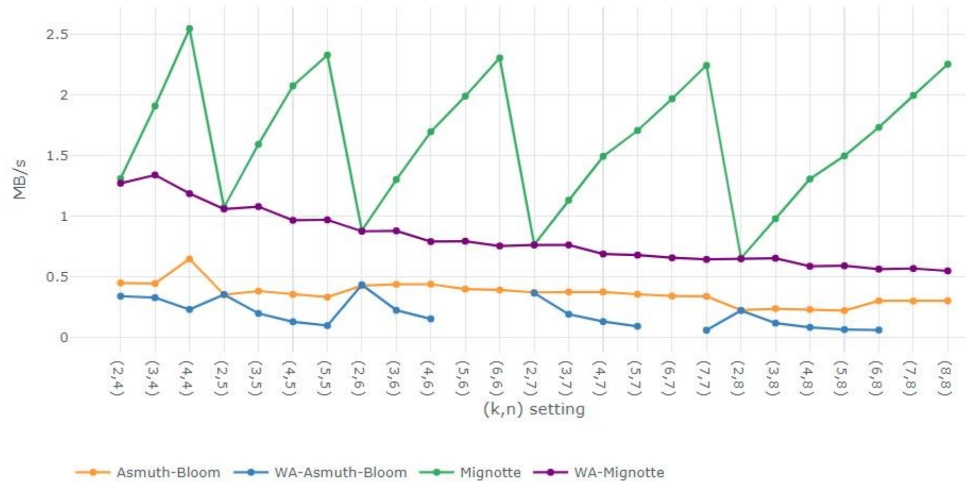


Figure 43. Storing speed of (n_v, k, N)-WA-RRNS and (k, n)-Systems

In Figure 44, we show the Extraction speeds (V_{ex}). Once again there is a decrement on the speeds, when we retrieve the shares from the clouds. We can see that for access structures, where $k = 2$, the shares have the same sizes in (n_v, k, N)-WA-RRNS and (k, n)-System, obtaining almost the same speeds.

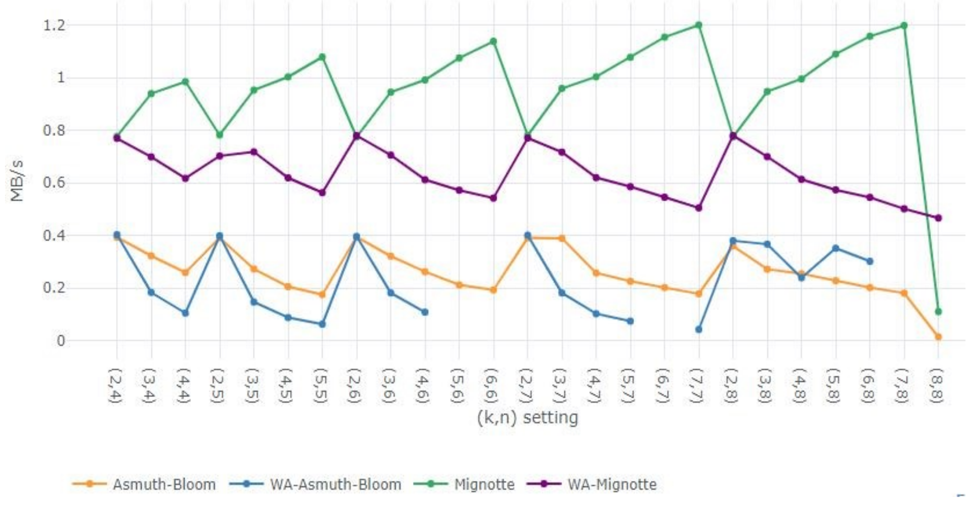


Figure 44. Extraction speed of WA-RRNS and (k, n) systems

When k increases (n_v, k, N)-WA-RRNS needs more shares to recover the data. Hence, when we compare decoding speeds and extraction speeds, we can see that on average, decoding speeds are 4x faster than the extraction speeds. When we consider communication to the clouds, the performance degradation, especially for (n_v, k, N)-WA-RRNS cannot be negligible.

4.2.4.3 Encoding and decoding performance of WA-RRNS with algorithm MRC-RRNS

In this section, we evaluate WA-RRNS applying the MRC-RRNS algorithm, named WA-MRC-RRNS. Figure 45 and Figure 46 depict the encoding and decoding speeds, respectively. We observe that MRC-RRNS outperforms WA-MRC-RRNS on both rates.

As expected, the encoding speed shows the same behavior as in the previous section. WA-MRC-RRNS system's encoding speed decreases while N increases. For example, Figure 45 shows that in setting (4,6), WA-MRC-RRNS is 9x faster than WA-Mignotte, almost 4x faster than Mignotte, and 0.472x slower than MRC-RRNS. WA-MRC-RRNS reaches a speed of 13.70 MB/s, MRC-RRNS has a rate of 29.03 MB/s, the speed of WA-Mignotte is nearly 1.52 MB/s while Mignotte is 3.523 MB/s; see Table 16.

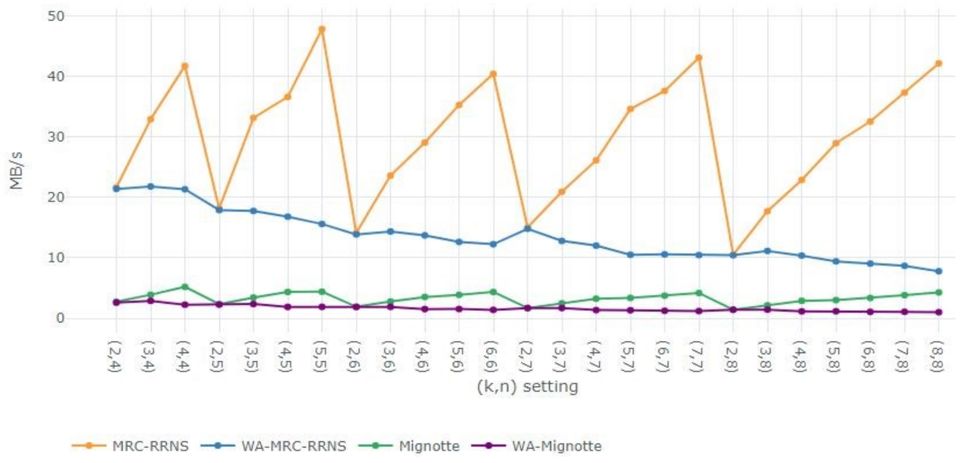


Figure 45. Encoding speeds of WA-MRC-RRNS and MRC-RRNS vs WA-Mignotte and Mignotte

Figure 46 shows the results of the decoding phase. On average, we can see that WA-MRC-RRNS is almost 10x faster than WA-Mignotte, and 0.879 slower than MRC-RRNS. For access structure (4,6), WA-MRC-RRNS reaches a speed of 29.54 MB/s, MRC-RRNS has a rate of 34.13 MB/s, the speed of WA-Mignotte is nearly 2.74 MB/s while Mignotte is 3.020 MB/s, see Table 17.

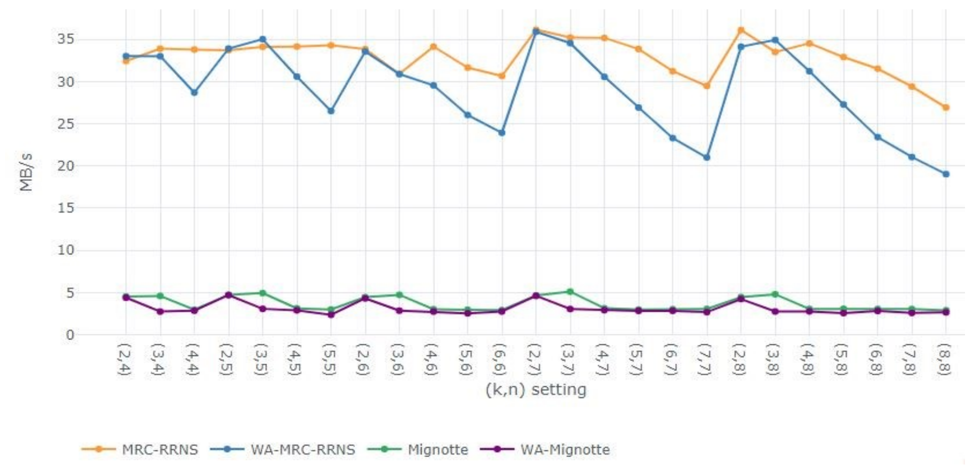


Figure 46. Decoding speeds of WA-MRC-RRNS and MRC-RRNS vs WA-Mignotte and Mignotte

Table 16. Encoding speed of Mignotte, MRC-RRNS, WA-Mignotte, and WA-MRC-RRNS

(K, N)	Encoding speed (MB/s)		(n_v, K, N)	n	Encoding speed (MB/s)	
	Mignotte	MRC-RRNS			WA-Mignotte	WA-MRC-RRNS
(2,4)	2.745970289	21.6309	(1,2,4)	4	2.583311806	21.3903
(3,4)	3.886664853	32.8947	(2,4,4)	8	2.881595251	21.7865
(4,4)	5.193995741	41.7014	(3,6,4)	12	2.250680831	21.3265
(2,5)	2.33018758	18.1653	(1,2,5)	5	2.268293789	17.8858
(3,5)	3.424422985	33.1235	(2,4,5)	10	2.359659265	17.7336
(4,5)	4.35502134	36.5898	(3,6,5)	15	1.876348626	16.7954
(5,5)	4.396763982	47.8011	(4,8,5)	20	1.890609343	15.5836
(2,6)	1.903637852	14.1123	(1,2,6)	6	1.876031817	13.8638
(3,6)	2.773002052	23.5849	(2,4,6)	12	1.893043067	14.3472
(4,6)	3.523484021	29.0360	(3,6,6)	18	1.522070015	13.7099
(5,6)	3.877922984	35.2485	(4,8,6)	24	1.534683855	12.6214
(6,6)	4.350852767	40.4040	(5,10,6)	30	1.393767074	12.2384
(2,7)	1.696583082	15.0127	(1,2,7)	7	1.686596617	14.7929
(3,7)	2.48188226	20.9161	(2,4,7)	14	1.688048616	12.7860
(4,7)	3.238866397	26.0960	(3,6,7)	21	1.359748718	12.0091
(5,7)	3.365870077	34.5901	(4,8,7)	28	1.323889588	10.4975
(6,7)	3.741254817	37.5798	(5,10,7)	35	1.240294694	10.5630
(7,7)	4.181126395	43.0848	(6,12,7)	42	1.194129659	10.4482
(2,8)	1.430144588	10.5152	(1,2,8)	8	1.411113933	10.4231
(3,8)	2.152667155	17.7022	(2,4,8)	16	1.432336427	11.1296
(4,8)	2.884670859	22.8623	(3,6,8)	24	1.145908534	10.3659
(5,8)	3.004175804	28.9435	(4,8,8)	32	1.160523628	9.4126
(6,8)	3.358183894	32.5203	(5,10,8)	40	1.058413861	9.0358
(7,8)	3.824238021	37.2995	(6,12,8)	48	1.076229323	8.6772
(8,8)	4.273504274	42.1230	(7,14,8)	56	1.008125491	7.7724

4.3 Performance analysis of a two level model (2L-RRNS and 2Lbp-RRNS)

In all previous sections, we analyzed the performance of the algorithms in a 1L-RRNS architecture. In this section, we will analyze their behavior on a two-level model (2L-RRNS and 2Lbp-RRNS). We do not evaluate the ability to detect and correct errors since there is no exact knowledge of the environment's variations to have a real representation of it and the performance of the schemes and models. As far as we know, this is the first work to evaluate the behavior of schemes based on RRNS in a two-level model in an experimental way. More so, in a dynamic environment like the Cloud. We evaluate the 2Lbp-RRNS performance in terms of encoding/decoding speeds using three algorithms: Mignotte, MRC8-RRNS, and MRC16-RRNS. We do not use a model with special moduli set for the experiments.

Therefore, let us recall that on a 2L-RRNS model, on the first level, we have n_1 moduli $p_{1,1}, p_{1,2}, \dots, p_{1,n_1}$ which are used to calculate shares $S_1, S_2, \dots, S_{n_1}$. On the second level, each S_i is transformed into the set of residuals $S_{i,j} = |S_i|_{p_{2,i,j}}$ by its own moduli set $p_{2,i,1}, p_{2,i,2}, \dots, p_{2,i,n_{2,i}}$, see Section 3.7.3 for more details.

4.3.1 Speed analysis of 2Lbp-RRNS

In the experiments, we studied the performance of a serial and parallel execution of 2L-RRNS. To draw all aspects of the proposed system, we run all settings (k_1, n_1) of Level 1 and all settings $(k_{2,i}, n_{2,i})$ of Level 2, where $1 \le i \le n_1$. For a better understanding, we will exemplify the results based only on the access structure (3,4) of Level 1.

Figure 47 and Figure 48 show examples of the encoding/decoding speeds, respectively, for up to eight clouds. On the first level, the access structure is limited to (k_1, n_1)= $(3,4)$. On the second level, we consider 27 variants of $(k_{2,i}, n_{2,i})$ from $n_{2,i} = 3$ to $n_{2,i} = 8$.

Due to the share size is decreasing while k_2 is increasing, the highest encoding speeds are obtained when $k_2 = n_2$. On the other hand, the lowest decoding speeds are obtained for $k_2 = n_2$ because it is necessary to decode all n_2 shares to recover the original data.

Table 17. Decoding speed of Mignotte, MRC-RRNS, WA-Mignotte, and WA-MRC-RRNS

(K,N)	Decoding speed (MB/s)		(n_v,K,N)	n	Decoding speed(MB/s)	
	Mignotte	MRC-RRNS			WA-Mignotte	WA-MRC-RRNS
(2,4)	4.534530449	32.42543637	(1,2,4)	4	4.389622931	33.00331544
(3,4)	4.596433168	33.88683354	(2,4,4)	8	2.747630169	32.99242685
(4,4)	2.95945546	33.78379925	(3,6,4)	12	2.848921683	28.6861864
(2,5)	4.736867036	33.70409362	(1,2,5)	5	4.691972036	33.8983206
(3,5)	4.933399112	34.10642762	(2,4,5)	10	3.067390571	35.01402163
(4,5)	3.132145206	34.1413608	(3,6,5)	15	2.892179547	30.58105375
(5,5)	2.993295019	34.28181206	(4,8,5)	20	2.363730913	26.50412027
(2,6)	4.456923831	33.84096304	(1,2,6)	6	4.30292599	33.55706234
(3,6)	4.728355951	30.89281612	(2,4,6)	12	2.854125639	30.87374059
(4,6)	3.020782987	34.12970846	(3,6,6)	18	2.744990393	29.54211101
(5,6)	2.964544053	31.64558411	(4,8,6)	24	2.539424566	26.03489867
(6,6)	2.925345191	30.65605327	(5,10,6)	30	2.752243078	23.92345593
(2,7)	4.664831833	36.12718417	(1,2,7)	7	4.605323754	35.89377092
(3,7)	5.10960094	35.21128372	(2,4,7)	14	3.049431281	34.54233015
(4,7)	3.135779241	35.16176012	(3,6,7)	21	2.910276185	30.55302346
(5,7)	2.984985523	33.85241902	(4,8,7)	28	2.826535515	26.91791273
(6,7)	3.031589159	31.22074486	(5,10,7)	35	2.834306445	23.2991719
(7,7)	3.058852319	29.47245676	(6,12,7)	42	2.678021478	20.99958962
(2,8)	4.469273743	36.10109956	(1,2,8)	8	4.233521019	34.11806407
(3,8)	4.792026069	33.48963355	(2,4,8)	16	2.758164166	34.9162171
(4,8)	3.056047919	34.50657204	(3,6,8)	24	2.772002772	31.22074486
(5,8)	3.080524921	32.8947519	(4,8,8)	32	2.565615619	27.27026064
(6,8)	3.056234719	31.51592996	(5,10,8)	40	2.819442878	23.41373116
(7,8)	3.045437934	29.39448729	(6,12,8)	48	2.582978174	21.04821001
(8,8)	2.904443799	26.92516041	(7,14,8)	56	2.653434871	19.02950443

We observe that Mignotte is the slowest of the considered algorithms, with speeds no higher than 1.23 MB/s for encoding and 1.3 MB/s for decoding. MRC8-RRNS has a maximum decoding speed of 10.02 MB/s in (2.5) of Level 2 and an encoding speed of 3 MB/s for settings where $k_2 = n_2$ of Level 2.

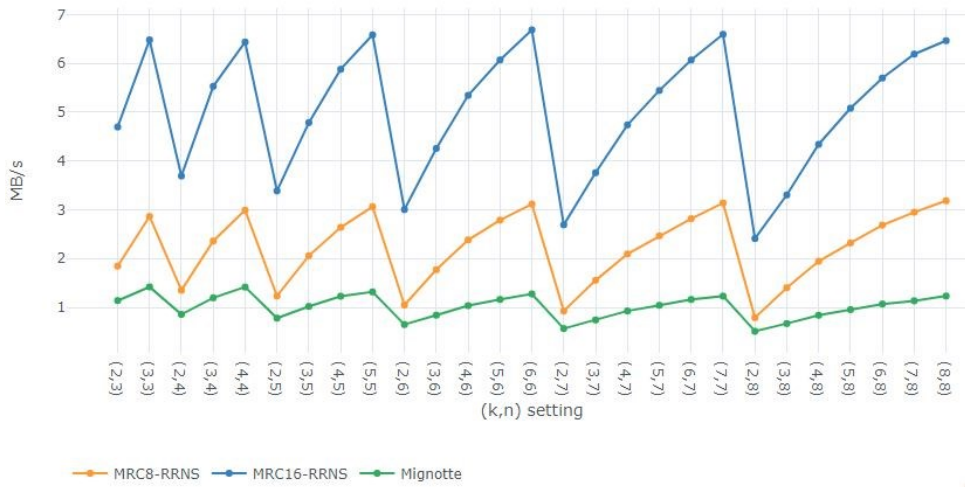


Figure 47. Encoding speeds for setting (3,4) on Level 1

We can see that MRC16-RRNS outperforms the two algorithms in all the experiments. For example, MRC16-RRNS achieves a maximum encoding speed of 6.68 MB/s for a setting (3,4) in Level 1 and (6,6) in Level 2. Finally, MRC16-RRNS is 2.53 times faster than MRC8-RRNS and 4.83 times faster than Mignotte for the encoding phase. In the decoding phase, MRC16-RRNS is 1.78 times faster than MRC8-RRNS and 11.43 times faster than Mignotte.

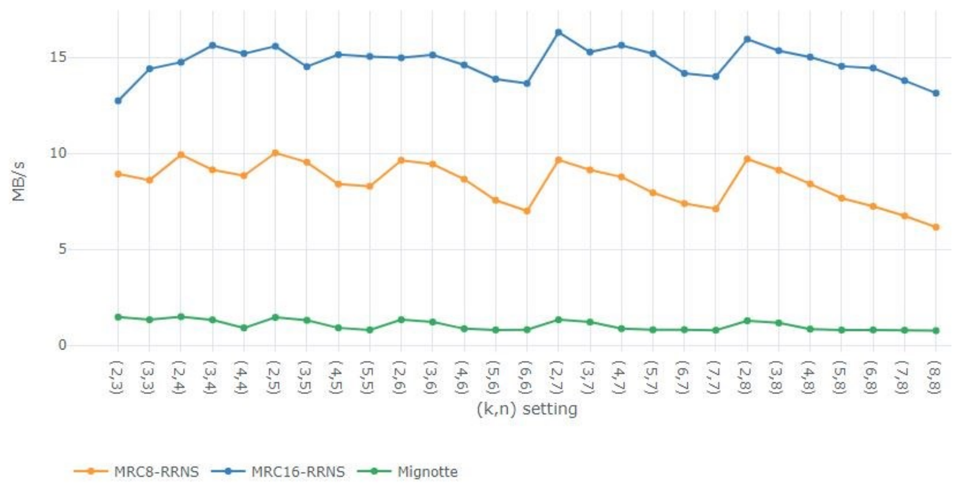


Figure 48. Decoding speeds for setting (3,4) on Level 1

Figure 49 and Figure 50 show boxplots for encoding/decoding speed, respectively. For each setting of Level 1, from $n_1 = 3$ to $n_1 = 8$, we execute all combinations of Level 2 settings. For statistical analysis, we run the experiments 30 times.

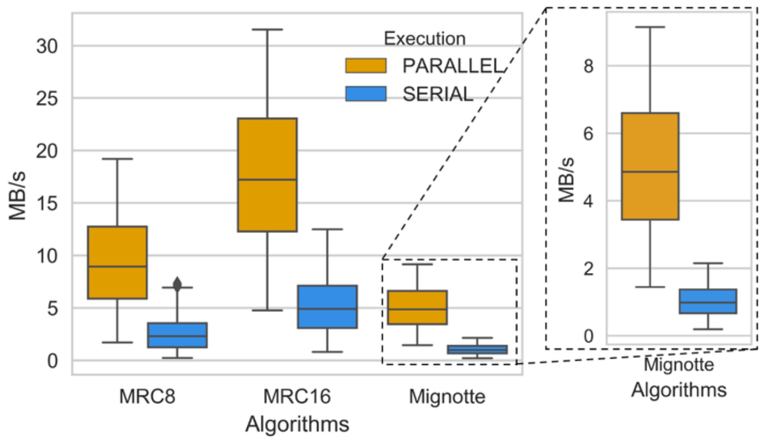


Figure 49. Boxplot encoding speeds for all combinations of settings

From the boxplot shown in Figure 49, we can draw the following conclusions about the encoding rate. MRC8-RRNS algorithm has outliers that show that there is a large gap between the minimum and maximum speeds. Most of the values are in the vicinity of the low-speed highlands.

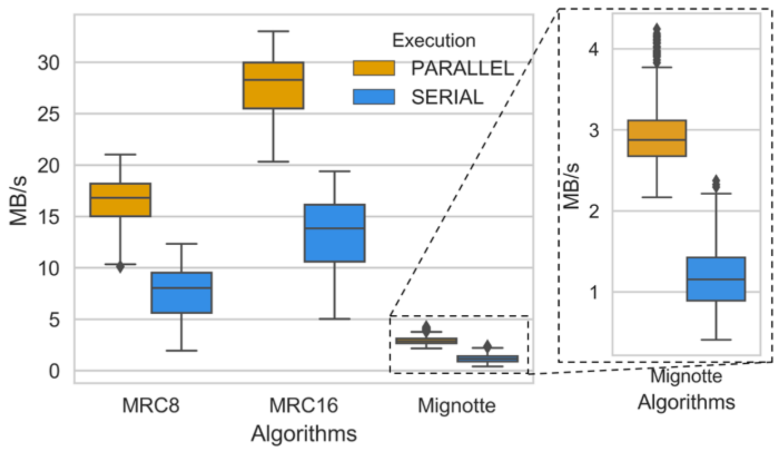


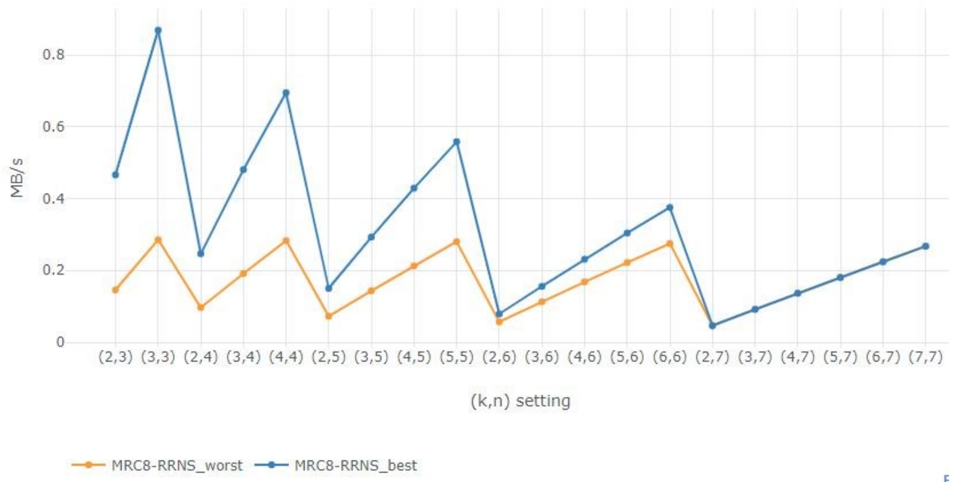
Figure 50. Boxplot decoding speeds for all combinations of settings

MRC16-RRNS algorithm is more balanced, and it is faster than MRC8-RRNS and Mignotte reaching a maximum speed higher than 30 MB/s for parallel execution. On the other hand, the decoding speed of MRC16-RRNS is higher than MRC8-RRNS and Mignotte (Figure 50). MRC16-RRNS and MRC8-RRNS have no outliers. Therefore, we can conclude that there are no speeds that stand out from the general sample. Mignotte has outliers, i.e., values that exceed the total sample's possible values, given that they are more than the median.

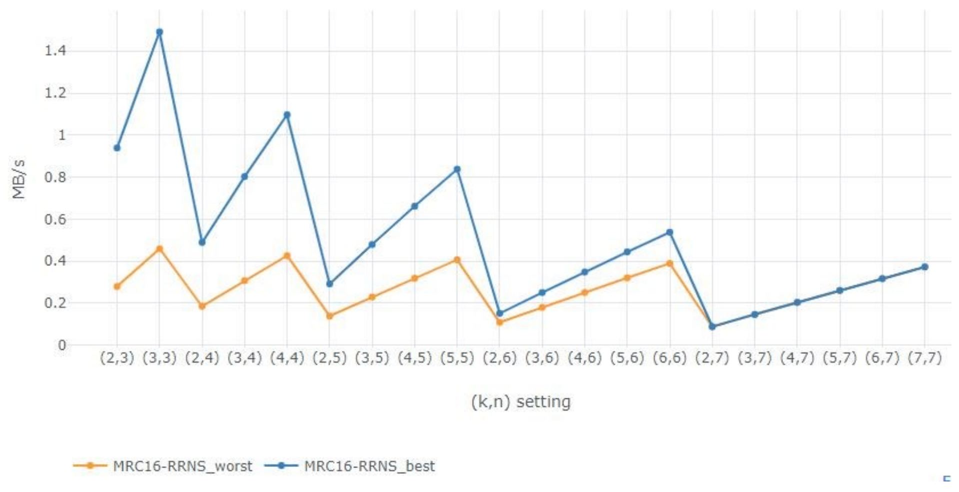
Overall, MRC16-RRNS shows the best performance for all combinations of settings of the 2L-RRNS.

4.3.1.1 Uploading and downloading performance for 2Lbp-RRNS

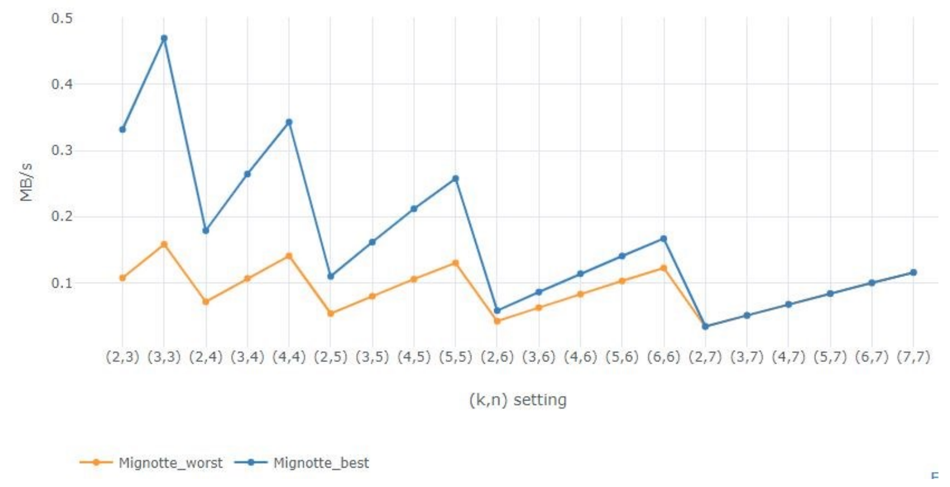
We want to know the behavior of the systems in a multi-cloud environment with a dynamic nature where parameters change over time and are difficult to predict and anticipate in advance. These types of non-stationarity are one of the main issues in the design of efficient algorithms capable of mitigating their consequences. So, we also realize a study of how a two level model will be affected by communications with the Cloud storage.



(a)



(b)



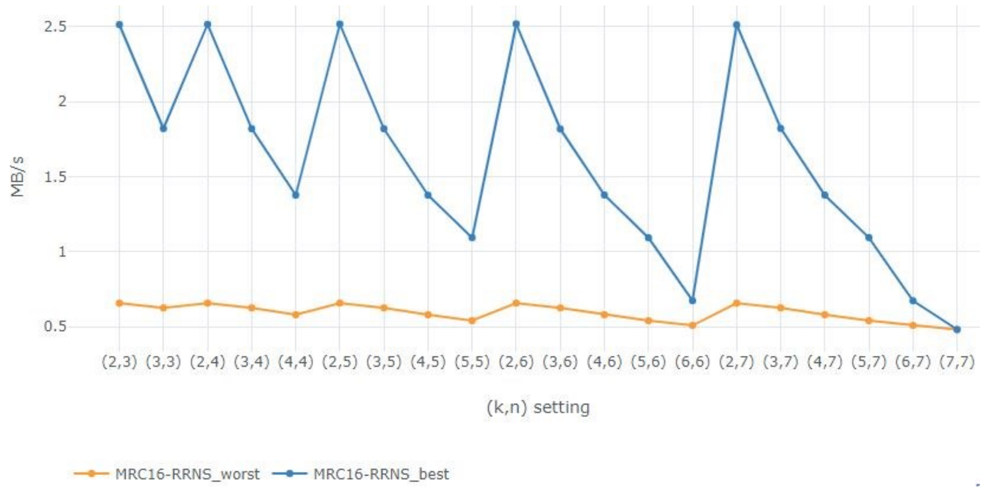
(c)

Figure 51. Storing speed for Level 1 setting (3,4). (a) MRC16-RRNS, (b) MRC8-RRNS, (c) Mignotte

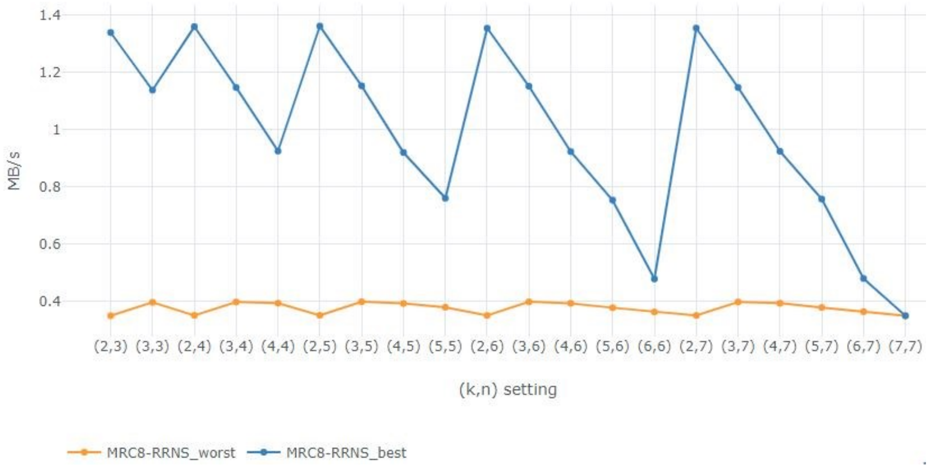
To determine the proposed scheme's practical applicability and study its properties, we consider the best and worst scenarios. In the best one, we select clouds with the best access speeds for data storage. In the worst one, we select the slowest clouds. Once again, for a better understanding, we will exemplify the results based only on the access structure (3,4) of Level 1 and (5,5) for Level 2 and the access speeds of Section 3.2.

Figure 51 shows the similar behavior of the three algorithms: (a) MRC16-RRNS, (b) MRC8-RRNS, (c) Mignotte. MRC16-RRNS has the storing speed V_u=0.837 MB/s, in the best case, and $V_u = 0.406$ MB/s, in the worst-case. MRC8-RRNS has storing speed V_u=0.558 MB/s, in the best case, and $V_u = 0.280$ MB/s, in the worst case. In the case of Mignotte, it has the storing speed V_u=0.257 MB/s, in the best case, and $V_u = 0.130$MB/s, in the worst case. For the storing speed, an average best scenario is 1.8x faster than the worst one for the three algorithms.

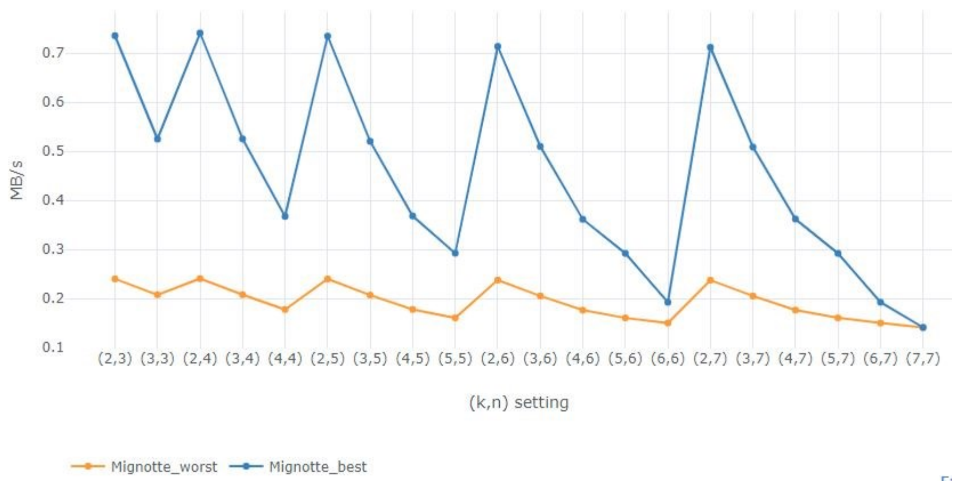
Figure 52 shows the extraction speeds of three algorithms: (a) MRC16-RRNS, (b) MRC8-RRNS, (c) Mignotte. It shows that MRC16-RRNS has the extraction speed V_d=1.093 MB/s, in the best case, and $V_d = 0.540$MB/s, in the worst case. MRC8-RRNS's best extraction speed is 0.760 MB/s, and its worst speed is 0.380 MB/s. Mignotte has $V_d = 0.292$MB/s, in the best case and $V_d = 0.160$MB/s, in the worst case. For the extraction speeds, an average best scenario is 3x faster than the worst one of the three algorithms.



(a)



(b)



(c)

Figure 52. Extraction speeds for Level 1 setting (3,4). (a) MRC16-RRNS, (b) MRC8-RRNS, (c) Mignotte

Chapter 5. Conclusions and future work

5.1 Conclusions

- The thesis studies the Residue Number System by exploring its property to detect and correct errors.

- The main contributions are the following:

- We propose a two-level 2Lbp-RRNS scheme based on a Redundant Residue Number System with a backpropagation and Hamming distance mechanisms to increase reliability.

- We prove the upper bounds of the traditional threshold 2L-RRNS and our solution to estimate the number of detectable and correctable errors.

- We evaluate the performance of the secret sharing schemes based on 1L-RRNS, WA-RRNS, 2L-RRNS, and 2Lbp-RRNS architectures in a multi-cloud environment varying data access structures on the first and second levels.

- To validate the practical applicability of the proposed scheme and study its properties in the real environment, we design the experimental framework based on eleven Cloud storages: Google Drive, OneDrive, Dropbox, Box, Egnyte, ShareFile, SalesForce, Alibaba Cloud, Amazon Cloud Drive, Apple iCloud, and Azure Storage.

- We evaluate properties of Mignotte, Asmuth-Bloom, AR-RRNS, Reed-Solomon, WA-RRNS, MRC8, and MRC16 based on the Mixed-Radix system, Finite Ring Neuronal Network, and signed binary window method, considering redundancy, storing/extraction speeds, coding/decoding speeds, for the best and worst scenarios.

We proved that the 2Lbp-RRNS model increases the number detected and corrected of errors using fewer segments available than state-of-the-art approaches. The detection and correction capacity of 2Lbp-RRNS comes with the cost of increasing complexity. It creates several possible solutions that Chinese Remainder Theorem cannot verify, encodes them, and evaluate Hamming distance of new encoded data with erroneous one. Knowing the Hamming distance, we can detect a sufficient number of correct segments allowing recovery of the original information.

To validate 2Lbp-RRNS, we carried out a theoretical analysis of the upper bounds of the number of detectable and correctable errors of 2L-RRNS and 2Lbp-RRNS models. We show that 2Lbp-RRNS could detect up to 1.58x more errors and correct up to 3.37x than a traditional 2L-RRNS model.

As a case study, we considered a data storage system in a cloud environment. We focused on the evaluation of (k, n) RRNS schemes with different parameters. For the experiments, we implemented two classical schemes based on the Chinese Remainder Theorem: Mignotte, Asmuth-Bloom, AR-RRNS approach based on the approximate range of RRNS, and MRC-RRNS based on the Mixed Base Conversion. We evaluated the scheme performance based on the encoding and decoding speeds, saving and extraction speeds, and redundancy.

We compared our solution with the Reed-Solomon error correction code with an AES encryption layer. Based on the average performance degradation results, RS + AES is 51.8% faster than the MRC-RRNS implementation for encoding and 38.14% faster for decoding.

When we evaluated the WA-RRNS scheme, we observed that its speeds were very low compared to traditional RRNS models, such as Mignotte. From the redundancy point of view, Asmuth-Bloom presented the worst results, being impractical. However, WA-RRNS can be a good option in scenarios where the handled information is not large.

5.2 Future work

In this section, we present the future considerations to continue improving our solution.

- Evaluate the 2Lbp-RRNs model with dynamic variations of the Cloud's characteristics, system failures, and possible attacks to study how a configurable approach can mitigate the non-stationary uncertainty and provide a good compromise over the criteria.

- Provide a multi-objective comparison with state-of-the-art approaches based on erasure codes, regeneration codes for our 2Lbp-RRNS scheme.

- Evaluate the system when the number of shares and their distribution on the second level can be dynamically adjusted to cope with the situation in each storage and different user requirements.

References

Abu-Libdeh, H., Princehouse, L., & Weatherspoon, H. 2010. RACS: a case for cloud storage diversity. *Proceedings of the 1st ACM Symposium on Cloud Computing, SoCC '10*, 229–239. https://doi.org/10.1145/1807128.1807165

Aguilera, M. K., Janakiraman, R., & Xu, L. 2005. Using erasure codes efficiently for storage in a distributed system. In *Proceedings of the International Conference on Dependable Systems and Networks*, 336–345. https://doi.org/10.1109/DSN.2005.96

Ali, M., Bilal, K., Khan, S. U., Veeravalli, B., Li, K., & Zomaya, A. Y. 2018. DROPS: division and replication of data in cloud for optimal performance and security. In *IEEE Transactions on Cloud Computing*, ***6***(2), 303–315. https://doi.org/10.1109/TCC.2015.2400460

Ali, P., Kambiz, M., Mohammad, S. S., Shiva, T. E., & Mehdi, R. 2014. Fault-tolerant and information security in networks using multi-level redundant residue number system. *Research Journal of Recent Science*, ***3***(3), 89–22.

Aora, I., & Gupta, A. 2012. Cloud databases: a paradigm shift in databases. *International Journal of Computer Science Issues*, ***9*(**4), 77.

Asmuth, C., & Bloom, J. 1983. A modular approach to key safeguarding. *IEEE Transactions on Information Theory*, ***29***(2), 208–210. https://doi.org/10.1109/TIT.1983.1056651

Attas, D., & Batrafi, O. 2011. Efficient integrity checking technique for securing client data in cloud computing. *International Journal of Electrical & Computer Sciences IJECS-IJENS*, 11.

Bajard, J., Didier, L., & Hilaire, T. 2011. ρ-Direct form transposed and residue number systems for filter implementations. In *2011 IEEE 54th International Midwest Symposium on Circuits and Systems (MWSCAS)*, 1–4. https://doi.org/10.1109/MWSCAS.2011.6026263

Barati, A., Dehghan, M., & Movaghar, A. 2008. Improving fault tolerance in ad-hoc networks by ssing residue number system. *Journal of Applied Sciences*, ***8***, 3273–3278.

Barsi, F., & Maestrini, P. 1973. Error correcting properties of redundant residue number systems. *IEEE Transactions on Computers*, ***C–22***(3), 307–315. https://doi.org/10.1109/T-C.1973.223711

Bellare, M., & Rogaway, P. 1998. PSS: provably secure encoding method for digital signatures. In *IEEE P1363a: Provably secure signatures.*

Bessani, A., Correia, M., Quaresma, B., Andre, F., & Sousa, P. 2013. DepSky: dependable and secure storage in a cloud-of-clouds. *ACM Transactions on Storage*, ***9***(4), 1–33. https://doi.org/10.1145/2535929

Bhagwat, D., Pollack, K., Long, D. D. E., Thomas Schwarz, S. J., Miller, E. L., & Pâris, J. F. 2006. Providing high reliability in a minimum redundancy archival storage system. In *Proceedings - IEEE Computer Society's Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, MASCOTS*, 413–421. https://doi.org/10.1109/mascots.2006.42

Blakley, G. R. 1979. Safeguarding cryptographic keys. *1979 International Workshop on Managing Requirements Knowledge, MARK 1979*, 313–317. https://doi.org/10.1109/MARK.1979.8817296

Blakley, G. R., & Borosh, I. 1979. Rivest-Shamir-Adleman public key cryptosystems do not always conceal messages. *Computers & Mathematics with Applications*, ***5***(3), 169–178. https://doi.org/https://doi.org/10.1016/0898-1221(79)90039-7

Boneh, D., Lynn, B., & Shacham, H. 2001. Short signatures from the weil pairing. In C. Boyd (Ed.), *Advances in Cryptology --- ASIACRYPT 2001* (pp. 514–532). Springer Berlin Heidelberg.

Bowers, K. D., Juels, A., & Oprea, A. 2009. HAIL: a high-availability and integrity layer for cloud storage. In *Proceedings of the ACM Conference on Computer and Communications Security*, 187–198. https://doi.org/10.1145/1653662.1653686

Campobello, G., Leonardi, A., & Palazzo, S. 2012. Improving energy saving and reliability in wireless sensor networks using a simple CRT-based packet-forwarding solution. *IEEE/ACM Transactions on Networking*, ***20***(1), 191–205. https://doi.org/10.1109/TNET.2011.2158442

Carnahan, K. 2020. *Billing-as-a-Service for property casualty insurers*. Recoverd February 2020 from https://www.input1.com/resources/articles/introduction-to-billing-as-a-service

Celesti, A., Fazio, M., Villari, M., & Puliafito, A. 2016. Adding long-term availability, obfuscation, and encryption to multi-cloud storage systems. *Journal of Network and Computer Applications*, ***59***, 208–218. https://doi.org/https://doi.org/10.1016/j.jnca.2014.09.021

Chang, C. H., Molahosseini, A. S., Zarandi, A. A. E., & Tay, T. F. 2015. Residue number systems: a new paradigm to datapath optimization for low-power and high-performance digital signal processing applications. In *IEEE Circuits and Systems Magazine* (**Vol. 15**, Issue 4, pp. 26–44). Institute of Electrical and Electronics Engineers Inc. https://doi.org/10.1109/MCAS.2015.2484118

Chang, F., Dean, J., Ghemawat, S., Hsieh, W., Wallach, D., Burrows, M., Chandra, T., Fikes, A., & Gruber, R. 2008. Bigtable: a distributed storage system for structured data. *ACM Transactions on Computer Systems*, ***26***(2), 1–26. https://doi.org/10.1145/1365815.1365816

Chen, P. M., Lee, E. K., Gibson, G. A., Katz, R. H., & Patterson, D. A. 1994. RAID: high-performance, reliable secondary storage. *ACM Comput. Surv.*, ***26***(2), 145–185. https://doi.org/10.1145/176979.176981

Cheon, J. H., Han, K., Kim, A., Kim, M., & Song, Y. 2019. A full RNS variant of approximate homomorphic encryption. In C. Cid & M. J. Jacobson Jr. (Eds.), *Selected Areas in Cryptography -- SAC 2018* (pp. 347–368). Springer International Publishing.

Cheon, J. H., Han, K., Kim, A., Kim, M., & Song, Y. 2018. Bootstrapping for approximate homomorphic encryption. In J. B. Nielsen & V. Rijmen (Eds.), *Advances in Cryptology -- EUROCRYPT 2018* (pp. 360–384). Springer International Publishing.

Chervyakov, N., Babenko, M., Tchernykh, A., Kucherov, N., Miranda-López, V., & Cortés-Mendoza, J. 2019. AR-RRNS: configurable reliable distributed data storage systems for Internet of things to ensure security. *Future Generation Computer Systems*, ***92***, 1080–1092. https://doi.org/10.1016/j.future.2017.09.061

Chervyakov, N. I., Babenko, M. G., Lyakhov, P. A., & Lavrinenko, I. N. 2014. An approximate method for comparing modular numbers and its application to the ivision of numbers in residue number systems. *Cybernetics and Systems Analysis*, ***50***(6), 977–984. https://doi.org/10.1007/s10559-014-9689-2

Chervyakov, N., Molahosseini, A., Lyakhov, P., Babenko, M., & Deryabin, M. 2017. Residue-to-binary conversion for general moduli sets based on approximate chinese remainder theorem. *International Journal of Computer Mathematics*, ***94***(9), 1833–1849. https://doi.org/10.1080/00207160.2016.1247439

Chessa, S., Di Pietro, R., & Maestrini, P. 2004. Dependable and secure data storage in wireless ad-hoc networks: an assessment of DS2. In R. Battiti, M. Conti, & R. Lo Cigno (Eds.), *Wireless On-Demand Network Systems* (pp. 184–198). Springer Berlin Heidelberg.

CloudHarmony. 2015. *service status | CloudHarmony*. Recovered April 2018 from https://cloudharmony.com/status

Cloud Security Alliance (CSA). 2010. Top threats to cloud computing. *Security*, *March*, 1–14. Recovered August 2017 from https://cloudsecurityalliance.org/artifacts/top-threats-to-cloud-computing-egregious-eleven/

Daemen, J., & Rijmen, V. 2000. The block cipher Rijndael. In J.-J. Quisquater & B. Schneier (Eds.), *Smart Card Research and Applications* (pp. 277–284). Springer Berlin Heidelberg.

Dean, J., & Ghemawat, S. 2008. MapReduce: simplified data processing on large clusters. *Commun. ACM*, ***51***(1), 107–113. https://doi.org/10.1145/1327452.1327492

Dilley, J., Maggs, B., Parikh, J., Prokop, H., Sitaraman, R., & Weihl, B. 2002. Globally distributed content delivery. *IEEE Internet Computing*, ***6***(5), 50–58. https://doi.org/10.1109/MIC.2002.1036038

Dimakis, A. G., Godfrey, P. B., Wu, Y., Wainwright, M. J., & Ramchandran, K. 2010. Network coding for distributed storage systems. *IEEE Transactions on Information Theory*, ***56***(9), 4539–4551. https://doi.org/10.1109/TIT.2010.2054295

Fatt Tay, T., & Chang, C. H. 2016. A non-iterative multiple residue digit error detection and correction algorithm in RRNS. *IEEE Transactions on Computers*, ***65***(2), 396–408. https://doi.org/10.1109/TC.2015.2435773

Forney, D. 2003. *Lecture notes, Principles of digital communication II* (pp. 101–115). Massachusetts Institute of Technology. Recovered June 2019 from https://dspace.mit.edu/bitstream/handle/1721.1/36834/6-451Spring-2003/OcwWeb/Electrical-Engineering-and-Computer-Science/6-451Spring-2003/LectureNotes/index.htm

Galletta, A., Fazio, M., Celesti, A., & Villari, M. 2020. Verifiable secret share for file storage with cheater identification. In *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*, 788–793. https://doi.org/10.1109/CCGrid49817.2020.000-9

García-Hernández, L. E., Tchernykh, A., Miranda-López, V., Babenko, M., Avetisyan, A., Rivera-Rodriguez, R., Radchenko, G., Barrios-Hernández, C. J., Castro, H., & Drozdov, A. Y. 2020. Multi-objective configuration of a secured distributed cloud data storage. In J. Crespo-Mariño & E. Meneses-Rojas (Eds.), *High Performance Computing. CARLA 2019. Communications in Computer and Information Science*. Springer, Cham. https://doi.org/https://doi.org/10.1007/978-3-030-41005-6_6

Garner, H. L. 1959. The residue number system. *Proceedings of the Western Joint Computer Conference, IRE-AIEE-ACM 1959*, 146–153. https://doi.org/10.1145/1457838.1457864

Gentry, C. 2009. A fully homomorphic encryption scheme [Stanford University]. In *Dissertation* (Issue September). http://cs.au.dk/~stm/local-cache/gentry-thesis.pdf

Gray, R. M. 2011. *Entropy and information theory*. Springer Science & Business Media.

Hamming, R. W. 1950. Error detecting and error correcting codes. *The Bell System Technical Journal*, ***29***(2), 147–160. https://doi.org/10.1002/j.1538-7305.1950.tb00463.x

Hashizume, K., Rosado, D. G., Fernández-Medina, E., & Fernandez, E. B. 2013. An analysis of security issues for cloud computing. *Journal of Internet Services and Applications*, ***4***(1), 1–13. https://doi.org/10.1186/1869-0238-4-5

Hema, V., & Durga, M. G. 2014. Data integrity checking based on residue number system and chinese remainder theorem in cloud. *International Journal of Innovative Research in Science Engineering and Technology*, ***3***(3), 2584–2588.

Herodotou, H., Lim, H., Luo, G., Borisov, N., Dong, L., Cetin, F. B., & Babu, S. 2011. Starfish: A self-tuning system for big data analytics. In *Proceedings of the Fifth Biennial Conference on Innovative Data Systems Research, CIDR*, 261–272.

Huang, C. H. 1983. A fully parallel mixed-radix onversion algorithm for residue number applications. *IEEE Transactions on Computers*, ***C–32***(4), 398–402. https://doi.org/10.1109/TC.1983.1676242

Irwin, L. 2020. *List of data breaches and cyber attacks in February 2020*. IT Governance UK. Recovered March 2020 fromwww.itgovernance.co.uk/blog/list-of-data-breaches-and-cyber-attacks-in-january-2020-1-5-billion-records-breached

Ito, M., Saito, A., & Nishizeki, T. 1989. Secret sharing scheme realizing general access structure. *Electronics and Communications in Japan*, ***72***(9), 56–64.

Jenkins, W. 1978. Techniques for residue-to-analog conversion for residue-encoded digital filters. *IEEE Transactions on Circuits and Systems*, ***25***(7), 555–562. https://doi.org/10.1109/TCS.1978.1084495

Jenkins, W., & Leon, B. 1977. The use of residue number systems in the design of finite impulse response digital filters. *IEEE Transactions on Circuits and Systems*, ***24***(4), 191–201. https://doi.org/10.1109/TCS.1977.1084321

Jia, W., & Wang, L. 2013. A unified unicast and multicast routing and forwarding algorithm for software-defined datacenter networks. *IEEE Journal on Selected Areas in Communications*, ***31***(12), 2646–2657. https://doi.org/10.1109/JSAC.2013.131206

Johnson, B. L., Palo, E. A., Cosentino, R. J., & Vaccaro, J. J. 1986. The residue number system for VLSI signal processing. In J. M. Speiser (Ed.), *Advanced Algorithms and Architectures for Signal Processing I* (**Vol. 0696**, pp. 176–187). SPIE. https://doi.org/10.1117/12.936891

Junior, J. A., Nascimiento, L. F., & Albini, L. C. 2011. Using the redundant residue number system to increase routing dependability on mobile ad-hoc networks. *Journal of Selected Areas in Telecommunications*, ***4***, 67–73.

Jyothi, G. N., Sanapala, K., & Vijayalakshmi, A. 2020. ASIC implementation of distribuited arithmetic based FIR filter using RNS for high speed DSP systems. *International Journal of Speec Tchnology*, 1–6.

Kar, A., Sur, K., Godara, S., Basak, S., Mukherjee, D., Sukla, A. S., Das, R., & Choudhury, R. 2016. Secuirity in cloud storage: an enhanced technique of data storage in cloud using RNS. In *2016 IEEE 7th Annual Ubiquitous Computing, Electronics Mobile Communication Conference (UEMCON)*, 1–4. https://doi.org/10.1109/UEMCON.2016.7777905

Karnin, E. D., Greene, J. W., & Hellman, M. E. 1983. On secret sharing systems. *IEEE Transactions on Information Theory*, ***29***(1), 35–41. https://doi.org/10.1109/TIT.1983.1056621

Krishna, H., Lin, K.-., & Sun, J.-. 1992. A coding theory approach to error control in redundant residue number systems. theory and single error correction. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, ***39***(1), 8–17. https://doi.org/10.1109/82.204106

Kulkarni, G., Sutar, R., & Gambhir, J. 2012. Cloud computing-storage as service. *International Journal of Engineering Research and Application (IJERA)*.

Lamehamedi, H., & Szymanski, B. K. 2007. Decentralized data management framework for data grids. *Future Generation Computer Systems*, ***23***(1), 109–115. https://doi.org/10.1016/j.future.2006.06.005

Leavitt, N. 2010. Will NoSQL databases live up to their promise? *Computer*, ***43***(2), 12–14. https://doi.org/10.1109/MC.2010.58

Li, W., & Ping, L. 2009. Trust model to enhance security and interoperability of cloud environment. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *5931 LNCS*, 69–79. https://doi.org/10.1007/978-3-642-10665-1_7

Lillard, T. V., Garrison, C. P., Schiller, C. A., Steele, J., & Murray, J. 2010. *Digital forensics for network, internet, and cloud computing*. Burlington, MA: Syngress Publishing.

Lin, S., & Costello, D. 1983. *Error control coding. Fundamentals and Applications*. Prentice-Hall.

Lopez-Falcon, E., Miranda-Lopez, V. Tchernykh, A., Babenko, M., & Avetisyan, A. 2019. Bi-objective analysis of an adaptive secure data storage in a multi-cloud. *Commun. Comput. Inf. Sci.*, 307–321.

Lu, M. 2004. *Arithmetic and logic in computer systems*. John Wiley & Sons.

MacWilliams, F. J., & Sloane, N. J. 1977. *The theory of error correcting codes* (3er ed.). North-Holland.

Menezes, A. J., Katz, J., Van Oorschot, P. C., & Vanstone, S. A. 2018. *Handbook of applied cryptography*. CRC press.

Mignotte, M. 1983. How to share a secret. In T. Beth (Ed.), *Cryptography* (pp. 371–375). Springer Berlin Heidelberg.

Miranda-López, V., Tchernykh, A., Cortés-Mendoza, J. M., Babenko, M., Radchenko, G., Nesmachnow, S., & Du, Z. 2018. Experimental analysis of secret sharing schemes for cloud storage based on RNS. In E. Mocskos & S. Nesmachnow (Eds.), *High Performance Computing* (pp. 370–383). Springer International Publishing.

Mora, A. C., Chen, Y., Fuchs, A., Lane, A., R., L., & Manadhata, P. 2012. *Top ten big data security and privacy challenges*. 2012 Cloud Security Alliance

Morelos-Zaragoza, R. H. 2006. *The art of error correcting coding*. John Wiley & Sons.

Mullen, G., & Panario, D. 2013. *Handbook of finite fields*. Chapman and Hall/CRC. https://doi.org/https://doi.org/10.1201/b15006

Nazarov, A., Chervyakov, N., Tchernykh, A., & Babenko, M. 2018. Reliability improvement of information systems by residue number system code. *International Journal of Combinational Optmization Problems and Informatics*, ***9***(1), 81.

Okman, L., Gal-Oz, N., Gonen, Y., Gudes, E., & Abramov, J. 2011. Security issues in NoSQL databases. In *2011IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications*, 541–547. https://doi.org/10.1109/TrustCom.2011.70

Omondi, A., & Premkumar, B. 2007. *Residue number systems. Theory and Implementation*. Imperial College Press.

Ozsu, M. T., & Valduriez, P. 1991. Distributed database systems: where are we now? *Computer*, ***24***(8), 68–78. https://doi.org/10.1109/2.84879

Parhami, B. 1990. Generalized signed-digit number systems: a unifying framework for redundant number representations. *IEEE Transactions on Computers*, ***39***(1), 89–98. https://doi.org/10.1109/12.46283

Pawan, N., Abhidnya, J., & Alexander, S. 2012. Data migration using active cloud engine. In *IEEE Cloud Computing for Emerging Markets, CCEM 2012 - Proceedings*, 68–71. https://doi.org/10.1109/CCEM.2012.6354605

Phong, L. T., Aono, Y., Hayashi, T., Wang, L., & Moriai, S. 2018. Privacy-preserving deep learning via additively homomorphic encryption. *IEEE Transactions on Information Forensics and Security*, ***13***(5), 1333–1345. https://doi.org/10.1109/TIFS.2017.2787987

Piestrak, S. J. 1994. Design of residue generators and multioperand modular adders using carry-save adders. *IEEE Transactions on Computers*, ***43***(1), 68–77. https://doi.org/10.1109/12.250610

Plank, J., Luo, J., Schuman, C. D., Xu, L., & Wilcox-O'Hearn, Z. 2009. A performance evaluation and examination of open-source erasure coding libraries for storage. *Fast*, ***9***, 253–265. https://www.usenix.org/legacy/event/fast09/tech/full_papers/plank/plank_html/

Plank, J., Simmerman, S., & Schuman, C. D. 2007. *Jerasure: a library in C/C++ facilitating erasure coding for storage applications*.

Pontarelli, S., Cardarilli, G. C., Re, M., & Salsano, A. 2008. Totally fault tolerant RNS based FIR filters. In *2008 14th IEEE International On-Line Testing Symposium*, 192–194. https://doi.org/10.1109/IOLTS.2008.14

Rabin, M. O. 1979. *Digitalized signatures and public-key functions as intractable as factorization*.

Rabin, M. O. 1990. The information dispersal algorithm and its applications. In *Sequences* (pp. 406–419). Springer New York. https://doi.org/10.1007/978-1-4612-3352-7_32

Rao, R. V., & Selvamini, K. 2015. Data security challenges and its solutions in cloud computing. *Procedia Computer Science*, ***48***, 204–209.

Reed, I. S., & Solomon, G. 1960. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, ***8***(2), 300–304. https://doi.org/10.1137/0108018

Rivest, R., Shamir, A., & Adleman, L. 1978. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, ***21***(2), 120–126. https://doi.org/https://doi.org/10.1145/359340.359342

Rivest, Ronald, Shamir, A., & Adleman, L. 1978. On data systems and privacy homomorphisms. *Foundations of secure computation*, ***4****(11)*, 169–180.

Robling Denning, D. E. 1982. *Cryptography and data security*. Addison-Wesley Longman Publishing Co., Inc.

Sangroya, A., Kumar, S., Dhok, J., & Varma, V. 2010. Towards analyzing data security risks in cloud computing environments. *Communications in Computer and Information Science*, ***54***, 255–265. https://doi.org/10.1007/978-3-642-12035-0_25

Schinianakis, D. M., Fournaris, A. P., Michail, H. E., Kakarountas, A. P., & Stouraitis, T. 2009. An RNS implementation of an F_{p} elliptic curve point multiplier. *IEEE Transactions on Circuits and Systems I: Regular Papers*, ***56***(6), 1202–1213. https://doi.org/10.1109/TCSI.2008.2008507

Shamir, A. 1979. How to share a secret. *Communications of the ACM*, ***22***(11), 612–613. https://doi.org/10.1145/359168.359176

Shieh, M.-D., Chen, J.-H., Wu, H.-H., & Lin, W.-C. 2008. A new modular exponentiation architecture for efficient design of RSA cryptosystem. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, ***16***(9), 1151–1161. https://doi.org/10.1109/TVLSI.2008.2000524

Silver-Greenberg, J., Goldstein, M., & Perlroth, N. 2014. Jpmorgan chase hack affects 76 million households. *New York Times*, 2.

Singh, D. U. 2013. Error detection and correction using reed solomon codes. *Error Detection and Correction Using Reed Solomon Codes*, *3*.

Skavantzos, A., & Abdallah, M. 1999. Implementation issues of the two-level residue number system with pairs of conjugate moduli. *IEEE Transactions on Signal Processing*, ***47***(3), 826–838. https://doi.org/10.1109/78.747787

Sun, J., & Kirshna, H. 1992. A coding theory approach to error control in redundant residue number systems. II. Multiple error detection and correction. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, ***39***(1), 18–34. https://doi.org/10.1109/82.204107

Sun, Z., Zhang, Q., Li, Y., & Tan, Y. 2018. DPPDL: A dynamic partial-parallel data layout for green video surveillance storage. *IEEE Transactions on Circuits and Systems for Video Technology*, ***28***(1), 193–205. https://doi.org/10.1109/TCSVT.2016.2605045

Szabo, N., & Tanaka, R. 1967. *Residue arithmetic and its applications to computer technology*. McGraw-Hill.

Tchernykh, A., Babenko, M., Chervyakov, N., Miranda-López, V., Kuchukov, V., Cortés-Mendoza, J. M., & Avetisyan, A. 2018. AC-RRNS: anti-collusion secured data sharing scheme for cloud storage. *International Journal of Approximate Reasoning*, ***102***, 60–73.

Tchernykh, A., Babenko, M., Kuchukov, V., Miranda-López, V., Avetisyan, A., Rivera-Rodriguez, R., & Radchenko, G. 2019. Data reliability and redundancy optimization of a secure multi-cloud storage under uncertainty of errors and falsifications. In *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 565–572. https://doi.org/10.1109/IPDPSW.2019.00099

Tchernykh, A., Babenko, M., Miranda-López, V., Avetisyan, A., Drozdov, A. Y., Rivera-Rodriguez, R., Radchenko, G., & Du, Z. 2020. Scalable data storage design for nonstationary IoT environment with adaptive security and reliability. *IEEE Internet of Things Journal*, ***7***(10), 10171–10188. https://doi.org/10.1109/JIOT.2020.2981276

Tchernykh, A., Miranda-López, V., Babenko, M., Armenta-Cano, F., Radchenko, G., Drozdov, A. Y., & Avetisyan, A. 2019. Performance evaluation of secret sharing schemes with data recovery in secured and reliable heterogeneous multi-cloud storage. *Cluster Computing*, ***22***(4), 1173–1185.

Tchernykh, Andrei, Babenko, M., Miranda-Lopez, V., Drozdov, A. Y., & Avetisyan, A. 2018. WA-RRNS: reliable data storage system based on multi-cloud. In *Proceedings - 2018 IEEE 32nd International Parallel and Distributed Processing Symposium Workshops, IPDPSW 2018*, 666–673. https://doi.org/10.1109/IPDPSW.2018.00107

Timarchi, S., & Navi, K. 2007. Efficient class of redundant residue number system. In *2007 IEEE International Symposium on Intelligent Signal Processing*, 1–6. https://doi.org/10.1109/WISP.2007.4447506

Tsafrir, D., Etsion, Y., & Feitelson, D. G. 2007. Backfilling using system-generated predictions rather than user runtime estimates. *IEEE Transactions on Parallel and Distributed Systems*, ***18***(6), 789–803. https://doi.org/10.1109/TPDS.2007.70606

Vouk, M. 2008. Cloud computing–issues, research and implementations. *Journal of Computing and Information Technology*, ***16***(4), 235–246.

Vukoli, M. 2010. The byzantine empire in the intercloud. *ACM SIGACT News*, ***41***(3), 105. https://doi.org/10.1145/1855118.1855137

Wang, X., & Yu, H. 2005. How to break MD5 and other hash functions. In R. Cramer (Ed.), *Advances in Cryptology -- EUROCRYPT 2005* (pp. 19–35). Springer Berlin Heidelberg.

Watson, R. W., & Hastings, C. W. 1966. Self-checked computation using residue arithmetic. *Proceedings of the IEEE*, ***54***(12), 1920–1931. https://doi.org/10.1109/PROC.1966.5275

Wei, Q., Veeravalli, B., Gong, B., Zeng, L., & Feng, D. 2010. CDRM: A cost-effective dynamic replication management scheme for cloud storage cluster. In *Proceedings - IEEE International Conference on Cluster Computing, ICCC*, 188–196. https://doi.org/10.1109/CLUSTER.2010.24

Weiss, N. E., & Miller, R. S. 2015. The target andother financial data breaches: Frequently asked questions. *Congressional Research Service*, 4.

Wu, C.-H., Hong, J.-H., & Wu, C.-W. 2001. RSA cryptosystem design based on the chinese remainder theorem. In *Proceedings of the 2001 Asia and South Pacific Design Automation Conference*, 391–395. https://doi.org/10.1145/370155.370419

Yau, S. S. S., & Liu, Y. C. 1973. Error correction in redundant residue number systems. *IEEE Transactions on Computers*, ***C–22***(1), 5–11. https://doi.org/10.1109/T-C.1973.223594

Yuke Wang. 2000. Residue-to-binary converters based on new Chinese remainder theorems. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, ***47***(3), 197–205. https://doi.org/10.1109/82.826745

Zhang, D., Jullien, G. A., & Miller, W. C. 1990. A neural-like network approach to finite ring computations. *IEEE Transactions on Circuits and Systems*, ***37***(8), 1048–1052. https://doi.org/10.1109/31.56084

Zhao, Y., Ou, K., Zeng, W., & Song, W. 2009. Research on cloud storage architecture and key technologies. In *ACM International Conference Proceeding Series*, ***403***, 1044–1048. https://doi.org/10.1145/1655925.1656114

Appendix

We describe the algorithms used for the analysis of a 2L-RRNS model of Section 3.7.3. For simplicity, in the algorithms, the variable $\tilde{S}$ represents an array with all the shares of level 2.

```
**2L-RRNS Encoding**
**Input**: settings $= (k_1, n_1), (k_{2,1}, n_{2,1}), \cdots, (k_{2,n_1}, n_{2,n_1})$
 S –Input data;
$\hat{p} = (p_{1,1}, \dots, p_{1,n_1}), (p_{2,1,1}, \dots, p_{2,1,n_{2,1}}), \cdots,$
$(p_{2,n_1,1}, \dots, p_{2,n_1,n_{2,n_1}})$. $W_{1,i} = (w_{1,i,0}, \dots, w_{1,i,l})$ and $W_{2,i,j} = (w_{2,i,j,0}, \dots, w_{2,i,j,l})$ is synaptic weights FRNN, algo – Mignotte, MRC8, MRC16
**Output**: $\tilde{S}$
1. Case algo == Mignotte:
   1.1 $\tilde{S} = Mignotte(\text{settings}, S, \hat{p}, \text{encoding})$
2. Case algo == MRC8:
   2.1 $\tilde{S} = MRC8(\text{settings}, S, \hat{p}, W_{1,i}, W_{2,i,j}, \text{encoding})$
3. Case algo == MRC16:
   3.1 $\tilde{S} = MRC16(\text{settings}, S, \hat{p}, W_{1,i}, W_{2,i,j}, \text{encoding})$
4. **return** $\tilde{S}$
```

```
**2L-RRNS decoding**
**Input**: settings $= (k_1, n_1), (k_{2,1}, n_{2,1}), \cdots, (k_{2,n_1}, n_{2,n_1})$
$\tilde{S}$–Representation of $S$ in 2L-RRNS;
$\hat{p} = (p_{1,1}, \dots, p_{1,n_1}), (p_{2,1,1}, \dots, p_{2,1,n_{2,1}}), \dots,$
$(p_{2,n_1,1}, \dots, p_{2,n_1,n_{2,n_1}})$
; $I_D$; $\widehat{W}_1 = (\overline{\widehat{w}}_{1,1}, \dots, \widehat{w}_{1,n_1})$ and $\widehat{W}_{2,i} = (\widehat{w}_{2,i,1}, \dots, \widehat{w}_{2,i,n_{2,i}})$ is synaptic weights DNN; algo – Mignotte, MRC8, MRC16
**Output**: S
1 Case algo == Mignotte:
   1.1 $S = Mignotte(\text{settings}, I_D, \tilde{S}, \hat{p}, \text{decoding})$
2 Case algo == MRC8:
   2.1 $S = MRC8(\text{settings}, I_D, \tilde{S}, \hat{p}, \widehat{W}_1, \widehat{W}_{2,i}, \text{decoding})$
3 Case algo == MRC16:
   3.1 $S = MRC16(\text{settings}, I_D, \tilde{S}, \hat{p}, \widehat{W}_1, \widehat{W}_{2,i}, \text{decoding})$
4. **return** $S$
```

Algorithms *Mignotte Encoding*, *MRC Encoding*, and the functions of algorithm 2L-RRNS encoding obtain the total shares $\tilde{S}$ from input data S. The three algorithms first convert S into n_1 shares, next take each share S_i at a time and convert it into $n_{2,i}$ shares for each combination of settings of Level 2.

```
**Mignotte Encoding**
**Input**: settings $= (k_1, n_1), (k_{2,1}, n_{2,1}), \cdots, (k_{2,n_1}, n_{2,n_1})$
 S –Input data;
$\hat{p} = (p_{1,1}, \dots, p_{1,n_1}), (p_{2,1,1}, \dots, p_{2,1,n_{2,1}}), \cdots,$
$(p_{2,n_1,1}, \dots, p_{2,n_1,n_{2,n_1}})$.
**Output**: $S_{i,j}$ - RNS encoded shares
1. For $i = 1$ to $n_1$ do:
   1.1 $S_i = |S|_{p_{1,i}}$
2. For $i = 1$ to $n_1$ do:
   2.1 For $j = 1$ to $n_{2,i}$ do:
        2.1.2 $S_2 = |S_i|_{p_{2,i,j}}$
3. **return** $\tilde{S}$
```

Mignotte Encoding uses mod operation $S_i = |S|_{p_i}$ to convert each input. *MRC* [8,16] *Encoding* represents the input into a residue representation, lines 1.1, and 2.1.2 using FRNN. To recover S, we use the *2L-RRNS Decoding* algorithm and functions *Mignotte Decoding* that uses classic CRT, and *MRC Decoding* that uses classic MRC decoding conversion (Huang, 1983) and FRNN modification based on FRNN (Zhang et al., 1990). The three functions use $k_{2,i}$ shares of Level 2 to retrieve the S_i shares of Level 1 for each combination of settings of Level 2. Next, we take $k_1$shares $S_i$and, finally, retrieve S.

```
**Mignotte Decoding**
**Input**: settings $= (k_1, n_1), (k_{2,1}, n_{2,1}), \cdots, (k_{2,n_1}, n_{2,n_1})$
$\tilde{S}$– Representation of S in 2L-RRNS;
$\hat{p} = (p_{1,1}, \dots, p_{1,n_1}), (p_{2,1,1}, \dots, p_{2,1,n_{2,1}}), \cdots,$
$(p_{2,n_1,1}, \dots, p_{2,n_1,n_{2,n_1}})$.
**Output**: $S$
1. $S_{list} = []; p_{list} = [];$ // auxiliary lists
2. For $i = 1$ to $k_1$ do:
   2.1. $j = I_D[i];$
   2.2. $S_j = CRTtoBin((S_{j,1}, \dots, S_{j,n_{2,i}}), (p_{2,j,1}, \dots, p_{2,j,n_{2,i}}));$
   2.3. $S_{list}.append(S_j); p_{list}.append(p_{1,j});$
3. $S = CRTtoBin(S_{list}, p_{list})$
4. **return** $S$
```

```
**MRC Encoding**
**Input**: settings $= (k_1, n_1), (k_{2,1}, n_{2,1}), \cdots, (k_{2,n_1}, n_{2,n_1})$
S –Input data;
$\hat{p} = (p_{1,1}, \dots, p_{1,n_1}), (p_{2,1,1}, \dots, p_{2,1,n_{2,1}}), \dots,$
$(p_{2,n_1,1}, \dots, p_{2,n_1,n_{2,n_1}})$
$W_{1,i} = (w_{1,i,0}, \dots, w_{1,i,l})$ and $W_{2,i,j} = (w_{2,i,j,0}, \dots, w_{2,i,j,l})$is synaptic weights FRNN,
**Output**: $S_{i,j}$
1. For $i = 1$ to $n_1$ do:
   1.1 $S_i = FRNN(S, p_{1,i}, W_{1,i})$
2. For i = 1 to $n_1$ do:
   2.1 For j = 1 to $n_{2,i}$ do:
       2.1.2 $S_{i,j} = FRNN(S_i, p_{2,i,j}, W_{2,i,j})$
3. **return** $S_{i,j}$
```

```
**MRC Decoding**
**Input**: settings $= (k_1, n_1), (k_{2,1}, n_{2,1}), \cdots, (k_{2,n_1}, n_{2,n_1})$
$\tilde{S}$ - Representation of $S$ in 2L-RRNS;
$\hat{p} = (p_{1,1}, \dots, p_{1,n_1}), (p_{2,1,1}, \dots, p_{2,1,n_{2,1}}), \dots,$
$(p_{2,n_1,1}, \dots, p_{2,n_1,n_{2,n_1}}); I_D; \widehat{W}_1 = (\bar{\widehat{w}}_{1,1}, \dots, \widehat{w}_{1,n_1})$ and $\widehat{W}_{2,i} = (\widehat{w}_{2,i,1}, \dots, \widehat{w}_{2,i,n_{2,i}})$ is synaptic weights DNN,
**Output**: $S$
1. $S_{list} = []; p_{list} = [];$ / auxiliary lists
2. For i = 1 to $k_1$ do:
   2.1. $c = DNN(S_{i,j}, p_{2,I_D[i],j}, \widehat{W}_{2,I_D[i]})$
   2.2. $S_{list}.append(S_i); p_{list}.append(p_{1,I_D})$
3. $S = DNN(S_{list}, p_{list}, \widehat{W}_1)$
4. **return** $S$
```

For *MRC Decoding*, we use a DNN. First, we calculate the coefficients (weights) of the neuronal architecture in lines from 2.1 to 3. Then, we perform the calculation of each part of the S_i for an FRNN.