Received March 27, 2019, accepted May 28, 2019, date of publication July 5, 2019, date of current version July 24, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2925735

Hy-SAIL: Hyper-Scalability, Availability and Integrity Layer for Cloud Storage Systems

DINO MACEDO AMARAL[1], JOÃO J. C. GONDIM[1], ROBSON DE OLIVEIRA ALBUQUERQUE[1], ANA LUCILA SANDOVAL OROZCO[2], AND LUIS JAVIER GARCÍA VILLALBA[2]

[1]PPEE Post Graduation in Electrical Engineering, Department of Electrical Engineering, University of Brasília, Brasília 70910-900, Brazil
[2]Group of Analysis, Security and Systems (GASS), Faculty of Computer Science and Engineering, Department of Software Engineering and Artificial Intelligence (DISIA), Universidad Complutense de Madrid (UCM), 28040 Madrid, Spain

Corresponding author: Luis Javier García Villalba (javiergv@fdi.ucm.es)

This work was supported in part by the European Union's Horizon 2020 Research and Innovation Programme under Grant 700326, in part by the Brazilian Research Council's CNPq under Grant 465741/2014-2 (INCT on Cybersecurity), CAPES under Grant 23038.007604/2014-69 FORTE, and FAP-DF under Grant 0193.001366/2016 UIoT, Grant 0193.001365/2016 SSDDC, and Grant Call 01/2019, in part by the Institutional Security Office of the Presidency of the Republic of Brazil under Grant 002/2017, in part by the LATITUDE/UnB Laboratory under Grant 23106.099441/2016-43 SDN, in part by the Ministry of the Economy under Grant 005/2016 DIPLA, Grant 011/2016 SEST, and Grant 083/2016 ENAP, in part by the EAGER: USBRCCR: Collaborative: Securing Networks in the Programmable Data Plane Era project funded by the NSF and RNP (Brazilian National Research Network), and in part by the Institutional Security Office of the Presidency of the Republic of Brazil under Grant 002/2017.

ABSTRACT Cloud storage has gained increasing attention from the industry and research communities with concerns about data stored in cloud computing environments, despite its many advantages. The reasons come from economic viability to the latency along with behavioral changes that may affect the applications that use data stored remotely. Among those challenges, there is the problem of ensuring the integrity and retrievability of users' data in the cloud. Several schemas have been proposed: proof of data possession (PDP), proof of retrievability (PoR), and proof of ownership (PoW) that differ on the approach to guarantee the effective retrieval of data stored remotely. In this paper, a novel PoR protocol is proposed: hyper-scalability, availability, and integrity layer (Hy-SAIL), where a new PoR scheme adds higher availability to data stored and a flexible manner to perform integrity checks. It is demonstrated that Hy-SAIL leads to an efficient and scalable cryptographic system that meets near-optimal bounds in terms of communication and storage complexities. It is also proposed a new adversarial model that aggregates the main functionalities of a realistic adversary in cloud computing environments. Hy-SAIL operates on data that are not affected by any kind of incremental change or update, which is the characteristic of various file types stored in clouds, such as stored video and audio streams. When compared to other systems, such as HAIL, Hy-SAIL is more scalable and efficient. The results collected with an unoptimized implementation of Hy-SAIL point to a better perspective than other approaches.

INDEX TERMS Cloud computing, cloud storage, proof of retrievability, message authentication code, homomorphic hashing, fountain codes, erasure codes.

I. INTRODUCTION

As technology aggregates and collaborates in global businesses, large enterprises have realized data center scalability is constrained by cost, subject to regulatory requirements and geographic limitations. Adding to those requirements, it is also mandatory to observe the way data center management evolved over time. Most methods used in the last decade do not provide the required agility that is now needed.

Recent technology has enabled the dissemination of cloud computing services and, among others, there is a specific area that take advantage over this new paradigm: cloud storage. So far, the main attractive feature of cloud storage is possibly the dramatic reduction in the cost of storing information with the increase in reliability, elasticity, scalability for data stored remotely. On this approach, even small companies will obtain large enterprise grade services inexpensively [2].

Under this new paradigm, customers relinquish physical control over their data and in return, obtain lower costs and highly-available resources. This, however, increases the

The associate editor coordinating the review of this manuscript and approving it for publication was Sabah Mohammed.

dependence over service providers and poses a number of new security concerns. One major concern in cloud computing is how to guarantee the integrity and availability of data remotely stored in cloud providers. This concern is not unrealistic since data loss events for leading service providers have already been reported [3].

In response, the cryptography community has been supplying new concepts as an answer to those demands. The leading ones are known as Proof of Retrievability (*PoR*) [1], [7]–[10], Proof of Data Possession (*PDP*) [4], Proof of Ownership (*PoW*) [5]. Though they have the same purpose, the difference lies in the ability of assuring recovery of stored data.

Current approaches for verifying file integrity (*PoR*, *PDP*, *PoW*) do not match an unbounded enquiry solution with low processing, storing and bandwidth cost. For each deployed approach, there is the need to assess which item will be sub-optimized in favor of a model deemed as acceptable. A limitation on all models so far is the challenge of rebuilding the file without the need to download entirely. The lack of elasticity when adding or removing storing units to abstraction layers created by the *PoR* models is another restriction that does not match with the "Cloud Computing" paradigm.

With those points observed, the question is which model should be deployed in *PoR* protocolos: unbounded or bounded. In an unbounded model, the user is not forced to limit the amount of times that an integrity check will run, however as shown in [8] this model requires a large amount of information about the file. In the bounded model, like in *HAIL* [1], the user computes and stores an amount of message authentication code of each file block and by some properties presented in homomorphic hash function, it will be possible to execute a reasonable amount of integrity checks.

Here, *Hy-SAIL* (*Hyper Scalability, Availability, Integrity Layer*) is presented, which focuses on two important concerns for storing data in cloud computing environments: availability and integrity. In order to provide availability, *Hy-SAIL* opted to use a flexible type of error correcting code, Online Codes [6] and distributing blocks generated by error correcting codes among different storage providers, that could be added at any moment. Regarding periodical integrity checks, "challenge-response" rounds are supported by algebraic properties explored in the polynomial representation of Galois Field with characteristic 2 ($GF(2^n)$), which provides the theoretical approach to compute an homomorphic *MAC* (Message Authentication Code) for any set of check blocks.

Hy-SAIL was conceived by the same PoR concept from *HAIL* [1], however the internal blocking and coding structure provides flexibility, scalability and availability with lower computational and communication cost. *Hy-SAIL's* scope was designed for data that does not change or update once stored remotely, like video and audio streams.

A. OUTLINE OF PAPER

This paper is structured as follows. In the following section, the key contributions are presented. In Section II related work is discussed. In Section III, the notation and polynomials algebra are presented. An overview of how *PoR* works is shown in IV. Then a brief explanation of how *Hy-SAIL* is built is described in Section V and a provably security approach concerning integrity and retrievability in Section VI. In Section VII, the details of each *Hy-SAIL*'s phase are explained. Finally, the implementation results are presented in Section VIII and conclusion in Section in IX.

B. STATEMENT OF CONTRIBUTION

The main contributions provided by *Hy-SAIL* may be summarized in the following aspects:

- **High scalability to storage providers**: *Hy-SAIL* satisfies elastic storage demands, due to usage of Online Codes [6] as error correcting codes, a special case of Fountain codes, which make it possible to add servers indistinctly to an abstraction layer built by *Hy-SAIL.*
- **Flexible *PoR* scheme**: In order to execute integrity checks, *Hy-SAIL* has both approaches, bounded and unbounded models, with low communication complexity, compared to other approaches. It focuses on the amounts of blocks not in storage providers and a complete verification on entire file in one single query, which also improves on previous work.
- **Heterogeneous environment support**: *Hy-SAIL* is designed to support heterogeneous systems providing storage capacity globally, as the abstraction layer encapsulates many restrictions found in other schemes. As a result, it is possible to have a storage layer with computing resources from providers, without any kind of constraints.
- **New adversarial model for distributed environment**: The adversarial model deployed for *Hy-SAIL* deals with realistic aspects found in the cloud computing approach. The attacker that tries to respond the challenges about *MAC* of given blocks has a negligible probability to be successful, in according to section VI.

II. PREVIOUS RELATED WORK

The *PoR* concept is defined in [7], where Juels *et al.* present a bounded model for verifying the integrity of files using "sentinels" across them, which optimize the usage of bandwidth in the "challenge-response" phase on the proposed *PoR*. In this model, "sentinels" are inserted in random positions, after the error correction code has been applied and, at the time of verification, they are asked whether the positions persist. If there is no relevant change on the "sentinels", it is deemed as retrievable. However, this approach may be compromised if some changes occur on the file that does not match the position of one of the "sentinels".

In [8], Schacham *et al.* propose two *PoR* frameworks with formal security definitions: the first has the security demonstrated using the bilinear pairing concept [15] in the Random Oracle model [16] and the second uses a pseudo-random function (PRF) and is secured at the standard model.

According to the same approach, Bowers *et al.* [10] propose a theoretical framework that supports a Byzantine adversarial model and enhances the Juels-Kalisky [7] and Schacham-Waters [8] models, limiting the adversary's error rate. Another approach in how to deal with an adversary is presented in [26], where the authors treats *PoR* schemes in the model of unconditional security, where an adversary has unlimited computational power.

A practical approach of a *PoR* model is shown in [1], with a more elaborated adversary model, where the attackers can corrupt servers and change file blocks during a pre-determined time round, the *HAIL* demands smaller computing and bandwidth than the *PoR*'s already proposed. Though it proposes to protect static elements, with minor modifications *HAIL* can meet the demands of assuring integrity of files with dynamic alterations. However, it still needs an unbounded query model for integrity checks, since it stores a limited quantity of authentication codes for files stored with their respective keys. In [24], authors show that is possible to obtain audit bandwidth that is independent of the data size.

On *PoR* frameworks, the task of auditing stored files is granted to a trustworthy third party to avoid possible storage processing overload on user stations [12]. In [23], authors present a scheme that is resilient against adversarial corruptions using simple cryptographic tools with very fast encoding/decoding times. With increasing offer of cloud storage vendors, it is very common to spread data among distinct providers. In [25] a study about Multi-prover Proof of Retrievability (*MPoR*) is presented concerning the confidentiality of the outsourced message. In [27], a notion called multi-proof-of-retrievability is introduced, which allows a verifier to check the availability of multiple files stored by a server in one single pass.

Another concept regarding storage on remote sites is defined in [4], where Ateniese *et al.* explain the Proof of Data Possession (*PDP*) model, which is a probabilistic proof that a trustworthy third party stores the file. In contrast with other models, the server accesses a small portion of the file to supply proof of its originality. This approach offers the advantage of keeping a small quantity of information about remotely stored data and a small amount of traffic generated for the challenge-response protocol. In [11], the *PDP* concept is applied for a hybrid approach, with public and private clouds storing data.

Halevi *et al.* [5] show the concept of Proof of Ownership (*PoW*), where there is a role inversion at the moment when data possession is being verified. The concern of this approach is to inhibit the attacker from downloading the file from the server, having only a portion of the knowledge regarding its information.

In [14] a scheme is presented with this gimmick, which in addition to locate which servers have corrupted files, also supports dynamic changes on files blocks, such as update, add and remove data. In [13], authors propose the combination of bilinear pairing signatures with *Merkle* trees in order to support public auditability and data dynamism.

III. BACKGROUND

A. NOTATION

In this subsection the notation used along this document is introduced:

- F - A file with arbitrary size that will be processed, distributed and stored by *Hy-SAIL*.
- n - File F is fragmented in n blocks for encoding, with the form: $F = \{m_i\}_{i=1}^{n} = \{m_1||m_2||\ldots||m_n\}$.
- m_i - Message block of file F with size $\frac{|F|}{n}$. These blocks will be used to compose check blocks c_j.
- $m_i(x)$ - A polynomial representation of message block (m_i) over GF(2^n), $m_i(x) = \sum_{i=1}^{n} a_{i-1}x^{i-1}$.
- c_j - Check blocks generated after encoding file F, which content is the result of bitwise XOR of random message blocks m_i's.
- δ - Loss rate that may occur when transmitted between 2(two) or more distinct nodes.
- ϵ - Redundancy rate that file F will be encoded to support some kind of failure.
- n' - Number of blocks after performing the first step in the error correcting codes, it is equal to $(1+k\delta)n$ blocks.
- D - Maximum degree that a unique check block may support. It is computed with parameters n, δ and ϵ. The formula is presented in equation 11.
- d - Degree of each check block c_j, which refers the quantity of message blocks m_i's that composes the check blocks.
- $\tau_{maximum}$ - Maximum number of tolerable corrupted check blocks in each cloud storage provider.
- $\tau_{provider}$ - Number of check blocks corrupted in each cloud storage provider.

B. POLYNOMIAL ALGEBRA AND GALOIS FIELD

Using well known facts of Polynomial Algebra [17], follows:

Definition 1: An irreducible polynomial $f(x) \in \mathbb{Z}_p[x]$ of degree m is said to be a primitive polynomial if x is a generator of $\mathbb{F}_{p^m}^{*}$, the multiplicative group of all the non-zero elements in $\mathbb{F}_{p^m} = \mathbb{Z}_p[x]/(f(x))$. [17]

Lemma 1: In [17], it is shown that the irreducible polynomial $f(x) \in \mathbb{Z}p[x]$ of degree m is called a primitive polynomial if and only if $f(x)$ divides $x^k - 1$ for $k = p^m - 1$ and for no smaller positive integer k.

In the light of lemma 1 and the fact that $f(x)$ is a k-degree primitive polynomial, we can conclude that $f(x)$ is irreducible. Also, the subset $\mathbb{Z}_2[x]/f(x)$ defines a Galois Field GF(2^k), so:

- Polynomial $f(x)$ is irreducible.
- The resulting set $\mathbb{Z}_2[x]/f(x)$ defines a Galois Field $GF(2^k)$.

IV. PROBLEM DESCRIPTION

As already mentioned, the use of cloud computing requires the guarantee of integrity and availability of data remotely stored in cloud providers. One possible approach to attack this problem is via a Proof of Retrievability Protocol (*PorR*) where the user places a challenge to the cloud provider concerning file availability and, possibly also, integrity; the provider then responds to the challenge; and the user decides whether or not to his satisfaction.

Proof of Retrievability protocols (*PoR*) were introduced in [7] by Ari Juels *et al.* and allow a storage provider to prove to a user that a file F previously stored is intact and the user may retrieve it with high probability. The challenge found in these protocols consists in storing the smallest amount of information about file F. That information will be used to perform periodical queries to the storage provider, which must answer to a user giving irrefutable proof that the original file is stored. The security of these protocols relies on the unforgeability of those queries and on the existence of a mechanism to retrieve the file F from any storage provider with overwhelming probability.

A basic model of *PoR* is a keyed hash approach of a single file, where the user possesses both the hash and the key. Prior to storing file F in a remote provider, the user computes and stores the hash $h_\kappa(F)$ with a random key κ. In order for the "challenge-response" protocol to be successful, the storage provider receives the key and must return a hash value of file F. For this type of approach, it seems obvious that a single verification is quite enough for an attacker to capture the keyed hash value, then modifies or deletes the content of original F and in the following queries, just responds the keyed hash value previously captured. For a practical and ideal *PoR* protocol, it is necessary a model in which the user performs an unbounded number of queries to a storage provider with a minimum computational cost.

A *PoR* protocol in its elementary form has three distinct phases: *Encoding*, *Challenge*, *Response*:

- **Encoding**: The user transforms file F into file F' after applying an error correcting code. File F' is stored at the cloud provider. This phase ensures that File F may be retrieved even if a fraction "α" of it is corrupted, where $0 < \alpha < 1$. After encoding, the size of an encoded file $|F'|$ is computed by the following formula: $|F/(1-\alpha)|$.
- **Challenge**: In order to audit the integrity of a file stored remotely, the user sends to storage provider t positions (p_1,p_3,...,p_t) of file F' that must be verified and a key ϱ to compute the keyed hash function in these positions. The storage provider captures the content in the requested positions from file F', Pos-F' = $F'[p_1]||F'[p_3]||\ldots||F'[p_t]$, computes the response, Resp = $hash_\varrho$(Pos-F') and sends back to user.
- **Response**: The user compares its own hash previously computed against the response sent by the storage provider. In this final step, it is possible to confirm the integrity of files stored remotely at any provider.

In a naive *PoR* scheme, the user may download the entire file "F'" and compare the hash with one which was previously computed. In another approach, the client may replicate the file three or four times across different storage providers and to check for integrity the client sends a challenge to each storage provider to compute a hash and compares the equality between them. Thus, in both approaches the optimization of computational items used is not a major concern (they are inherently inefficient - the first refers to bandwidth usage and second to computational and storage usage).

In this context, it is easy to notice that *PoR* protocols have to satisfy some criteria to make them practical for cloud computing industry: (1) an unbounded model that allows a user to perform as many queries as needed; (2) minimal computational and communication overhead for the "challenge-response" phase; (3) the storage provider must access some fraction of data to perform integrity checks, in the worst case the full copy of the entire file.

Using the criteria described above which provide the requirements for a realistic approach in terms of computational resources, the main purpose of *Hy-SAIL* is to supply a simple model for distributing data in a cloud computing environment preserving their integrity and assuring their retrievability. Essentially, *Hy-SAIL* has only two entities: user and cloud storage provider. Integrity is verified through a *PoR* mechanism that is initialized by user. In order to correctly respond to challenges submitted by the user, the cloud provider must possess in its storage units the original file contents. So, to assure the retrievability of original data, an *Online Codes* is implemented and the integrity is checked through a homomorphic *MAC*. The next section shows how these requirements might be satisfied.

V. *Hy-SAIL* OVERVIEW

The *Hy-SAIL* has 2 components: the protocol which corresponds to the code implemented to support all phases is described in VII, and the system that represents the architecture proposed. In figure 1 it is shown the cloud storage architecture proposed by *Hy-SAIL*, with the entities covered in a *PoR* scheme, and their respective attributions.

- **User:** In this context, the user must be authenticated and wishes to store files, in any format, in a cloud storage environment. The user is responsible for initializing the challenge to storage provider, in order to ensure that the stored files are available and have a minimum amount of loss, not compromising retrievability.
- **Storage Provider:** This entity supplies storage and processing services of a large amount of data in a distributed storing environment. Typically, these services are performed together in a distributed file system, such as HDFS (*Hadoop Distributed File System*) [20]. These computing resources are requested from different providers, in order to ensure a higher level of availability, retrievability and scalability.

In a cloud storage approach, one given provider may work collaboratively with other providers, that possess redundant servers scattered over the Internet without any technological restriction. The storage interface in the *Hy-SAIL* (figure 1) which may also work as gateway to other providers allows a high degree of interaction among different distributed file systems. This additional feature offers an effective way of

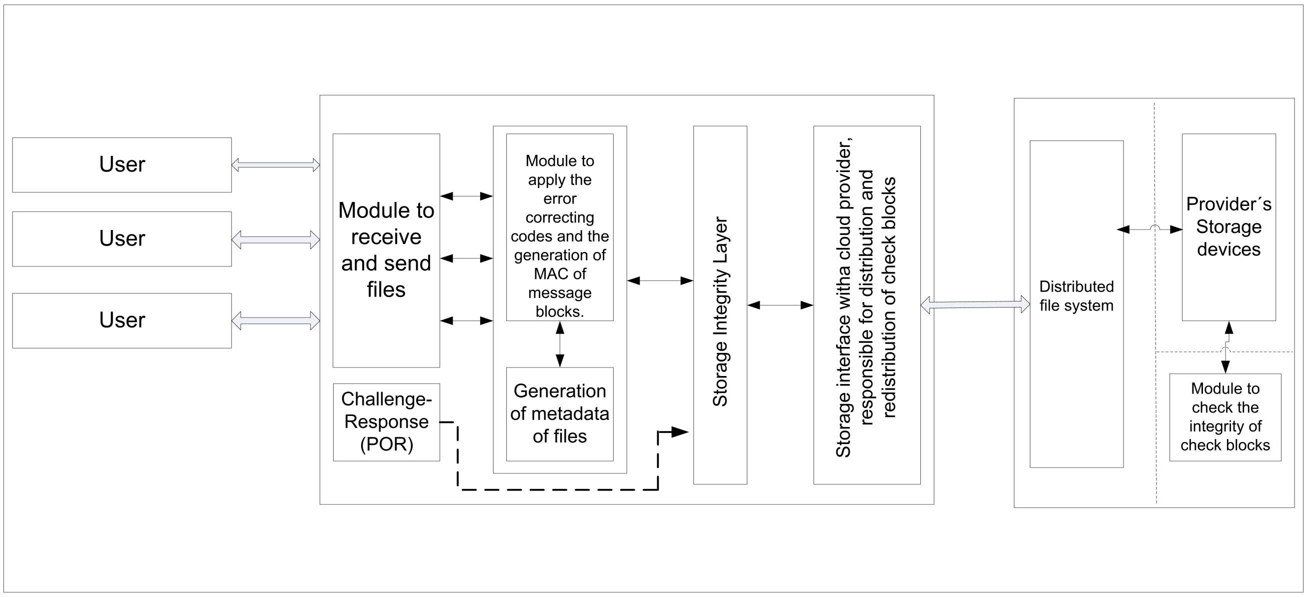
User
User
User
Module to receive and send files
Challenge-Response (POR)
Module to apply the error correcting codes and the generation of MAC of message blocks.
Generation of metadata of files
Storage Integrity Layer
Storage interface witha cloud provider, responsible for distribution and redistribution of check blocks
Distributed file system
Provider´s Storage devices
Module to check the integrity of check blocks

FIGURE 1. **An architecture proposed by Hy-SAIL.**

making simultaneous downloads from multiple sources and maximizes the use of providers' resources.

In addition to the possibility of redundancy by cloud providers, error correcting codes enable an additional perspective in order to ensure availability of stored content. With erasure codes [6], it is possible to convert a file of "n" blocks into an encoded message with "$(1 + \epsilon)n$" blocks, where $\epsilon > 0$, such that the original file will be recoverable with any given "n" blocks.

Upon receiving the parameters needed to encode a file, which is described in subsection VII-A, *Hy-SAIL* outputs check blocks $c_1, c_2, \ldots, c_j$. which are the bitwise XOR of message blocks m_i's (m_i's are chosen randomly from a set of "n" blocks). After generating these blocks, *Hy-SAIL* distributes them randomly among cloud providers, which provides a higher level of retrievability of files encoded by *Hy-SAIL*.

The paradigm of cloud computing has a dynamic environment, so it should be necessary to consider the possibility that any device may be compromised, turned-off or unplugged at any moment, which may imply unavailability of some check blocks. In order to mitigate this situation, Online Codes [6], error correcting codes implemented in *Hy-SAIL*, provide local encodability, which allows any check block of an encoded message to be computed independently [18]. This feature allows a redistribution mechanism (subsection VII-E) to reconstruct any check blocks without the need to download the whole file.

Hy-SAIL verifies the integrity of check blocks through properties of *Galois Field*. In [19], the interested reader may find a detailed explanation on how any element may be represented in some *Galois Field*. In a standard form, an element in the $GF(2^n)$ may be represented as a polynomial $a(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \ldots + a_1x + a_0$ of degree $n-1$ with coefficients belonging to $GF(2)$. This type of representation allows *Hy-SAIL* to compute the *MAC* of each message block m_i and is used to verify the integrity of check blocks.

In order to ensure the integrity of check blocks, a XOR homomorphic function is necessary, since the storage provider stores check blocks ($c_i = m_1 \oplus m_2$) and the user possesses only the *MAC's* of message blocks m_i's. It means that for any group of messages (ex.: m_1, m_2), a function (f) that builds the *MAC* for *Hy-SAIL* must be additive homomorphic satisfying:

$$f(m_1) \oplus f(m_2) = f(m_1 \oplus m_2). \tag{1}$$

The *MAC* proposed by *Hy-SAIL* allows flexibility in the *Auditing* phase as shown in subsection VII-D, enabling both the Bounded and the Unbounded model to perform integrity checks. To the best of the authors' knowledge, it was not found in literature a proposal with such approach, using *Online Codes* to encode the original message to improve the availability and make use of properties in the *Galois Field* to build a homomorphic *MAC* to verify the integrity which is provably secure, as shown in section VI. The communication and storage complexities make this protocol attractive and practical for distributing data in a cloud computing environment.

From a practical point of view, *Hy-SAIL* is organized in 5 phases: Encoding, Decoding, Auditing, Distribution, Redistribution. All these phases are explained in section VII.

A. BUILIDING A MAC FOR Hy-SAIL

The challenge for *PoR* protocols consists in storing a minimum amount of information about the file F to perform the integrity checks over the entire file F.

Firstly, each message block is represented in polynomial form over a Galois Field GF(2^k) 1, so the message blocks

m_1,m_2, ..., m_n, will be represented by the following polynomials $m_1(x)$,$m_2(x)$,..., $m_n(x)$, in the form $\sum_{i=1}^{t-1} a_i x^i$, where $a_i \in \{0, 1\}$.

In order to compute a *MAC* for each message block, *Hy-SAIL* takes the remainder of a polynomial division of each message block $m_i(x)$ against a set of primitive polynomials $P_j(x)$'s over $GF(2)$, so $MAC_{hysail} = m_i(x)$ mod $P_j(x)$. The arithmetic operations in $GF(2)$ supply the theoretical basis required to perform integrity checks in the *PoR* model used by *Hy-SAIL*. In order to execute the Auditing phase (section VII-D) in the *PoR* protocol, the user stores some *MAC's* for each message block and the corresponding challenges to each *MAC*.

As already known, blocks stored at cloud providers are the result of bitwise XOR over the contents of message blocks. However, the user has the remainder of the polynomial division between the content of message blocks and a challenge P_j. As exposed, it is clear that a function with XOR homomorphic property is necessary to verify the integrity of check blocks stored at cloud provider.

Assuming that a user has the authentication codes of m_a and m_b blocks against one of the challenges already generated. Resulting in $\sigma_{a,j}$ and $\sigma_{b,j}$, after computing the following equations:

$$\begin{aligned} m_a(x) \ mod \ P_j(x) &= \sigma_{a,j}(x) \\ m_b(x) \ mod \ P_j(x) &= \sigma_{b,j}(x) \end{aligned} \tag{2}$$

Suppose that a cloud storage provider has in its file systems the following check block: $c_e = m_a \oplus m_b$. The user sends to some provider a primitive polynomial from a set of challenges P_j's, and asks the remainder of a polynomial division as follows:

$$[m_a(x) \oplus m_b(x)] \ mod \ P_j(x) = \sigma_{ab,j}(x) \tag{3}$$

Since in $GF(2)$, the sum is equivalent to bitwise XOR, integrity will hold if the following equation is true:

$$\begin{aligned} &\left[m_a(x) \ mod \ P_j(x)\right] \oplus \left[m_b(x) \ mod \ P_j(x)\right] \\ &\qquad = [m_a(x) \oplus m_b(x)] \ mod \ P_j(x) \end{aligned} \tag{4}$$

In other words,

$$\sigma_{a,j}(x) \oplus \sigma_{b,j}(x) = \sigma_{ab,j}(x) \tag{5}$$

After this round, *Hy-SAIL* uses another challenge to proceed in the *PoR* protocol, repeatedly until completes the set of P_j challenges.

B. ADVERSARIAL MODEL

In a distributed storage environment, two important threats are relevant: a dishonest cloud provider and an adversary who aims at compromising the system. In the former, the cloud provider may wish to store an amount of data bigger than its total capacity (for example, by deleting rarely accessed files or hiding possible failures in the storage system). In the later, an adversarial entity may compromise a set of encoded blocks in a given time interval so that the retrievability of a file F is affected. So, it may be necessary to limit some parameters to avoid compromising *Hy-SAIL* availability.

Assume that *Hy-SAIL* is connected to a set of N storage providers, denoted by $\mathcal{S} = \{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_N\}$, and a trusted entity $\mathcal{T}$. Clearly, the trusted entity $\mathcal{T}$ refers to the user's machine, while the set $\mathcal{S}$ refers to the set of storage providers. It is assumed that a trusted entity $\mathcal{T}$ and each storage provider $\mathcal{S}_N$ are connected through a private and authenticated channel, such as Virtual Private Network.

Each storage provider contains in its setup two distinct elements: a *coding system* and a *storage system*. The coding system establishes the way the storage provider will reply to challenges, while the storage system comprises a distributed file segment.

In *Hy-SAIL* it is proposed a more general adversarial model, here called the *Bounded Corruption Model*. In this adversarial setting, the malicious entity is allowed to corrupt all the N storage providers simultaneously, as long as it affects only a bounded fraction of total check blocks. Thus, the malicious entity may perform arbitrary actions to corrupt check blocks stored in storage providers.

The description of the Bounded Corruption Model is as follows:

Definition 2 (Bounded Corruption Model): Let $\mathcal{C} = \{c_1, c_2, \ldots, c_t\}$ denote the set of generated check blocks, and $|t|$ the total amount of check blocks stored in the cloud. The adversary is able to corrupt up to $\alpha|t|$ check blocks in a given round, where $0 \leq \alpha < 1$. The adversary has the capability to perform bit flipping and erase actions in check blocks stored in each target server. α is called the *corruption density* of the system.

It is assumed that the adversary $\mathcal{A}$ acts in time cycles, called rounds. Each round comprises three phases:

1) The adversary gets access to an arbitrary set of storage providers and corrupts up to $(1 - \delta)(1 + k\delta)n$ check blocks of encoded file F', where $(1 - \delta)$ is loss effect that may corrupt some blocks and $(1 + k\delta)n$ is the amount of blocks considering some loss.
2) The entity $\mathcal{T}$ performs an auditing process so it can verify the integrity of check blocks.
3) If $\mathcal{T}$ detects any corruption in the auditing process, it may perform a redistribution of check blocks so that the original file can be retrieved.

VI. PROOF OF SECURITY

The proof of security involves the aspects addressed in *Hy-SAIL*, namely integrity and availability. Initially, it is proved that the probability of forging the *MAC* of selected check blocks is negligible, and so integrity is preserved. Then, availability is also established showing that a file is successfully retrieved with overwhelming probability. These results are then combined, proving the *Hy-SAIL* is secure against file corruption.

In this section specifically, our goals are: 1) prove that the *MAC* proposed by *Hy-SAIL* is 2-Universal Hash Function

(2-UHF) [21], which is defined bellow, and consequently any attacker that compromises a cloud provider has negligible probability to answer any challenge correctly; 2) limit the adversarial advantage, defining a number of check blocks that may be corrupted.

A. INTEGRITY PROOF

Theorem 1: The MAC_{hysail} is a 2-Universal Hash Function.

It follows the definition of 2-UHF (2-Universal Hash Function):

Definition 3 (2-Universal Hash Functions [21]): Let $\mathcal{U} = \{0, 1\}^*$ denote the set of all the binary finite strings and let $\mathcal{T} = \{0, 1\}^\lambda$ denote the set of all λ-bit strings. Let $h : \mathcal{U} \rightarrow \mathcal{T}$ denote a hash function. A family $\mathcal{H}$ of hash functions h is said to be 2-universal if:

$$\Pr[h(x_1) = h(x_2)|\ \ h \in_R \mathcal{H}] \leq 2^{-\lambda},\ \forall x_1 \neq x_2 \in \mathcal{U} \quad (6)$$

Proof: In the first step, it is necessary to know the size of $\mathcal{H}$, which corresponds to all primitive polynomials with degree at most "λ " in $GF(2^\lambda)[x]$. Facts mentioned in III-B are needed to support Theorem 1.

MAC_{hysail} maps values to a set $\mathcal{T}$, through a family $\mathcal{H}$ with hash functions "h". In the case of *Hy-SAIL*, the remainder of polynomial division has elements in the set $\mathcal{T}$, with all elements with polynomial representation in $GF(2^\lambda)$.

Hash functions $h \in \mathcal{H}$ have a key property: the polynomials $P(x)$ are uniformly taken from "h", the set of λ-degree primitive polynomials. The set $\mathcal{H}$ has $(2^\lambda - 1)/\lambda$ such polynomials. As an example, in *Hy-SAIL* the family $\mathcal{H}$ has approximately 2^{120} distinct hash functions for $\lambda = 127$.

According to the definition of congruency properties [17], each polynomial $m_i(x) \in \mathbb{Z}_2[x]$ is congruent to a single element of $GF(2^\lambda)$. We know that the residue of the division of a polynomial by another over a given field is unique. Consequently, the congruency between two distinct polynomials will occur with probability $|\mathbb{Z}_2[x]/p(x)|^{-1} = |GF(2^\lambda)|^{-1} = 2^{-\lambda}$. So, it is possible to state that the probability that two distinct elements have the same hash value is the inverse of the cardinality of the result image, namely $1/M$. As posted before, "M" has 127 bits in its size, so:

$$Pr_{h \in \mathcal{H}}[m_1(x) \bmod p(x) \equiv m_2(x) \bmod p(x)] \leq 1/M(2^{-127}) \quad (7)$$

Therefore, according to equation 7, the MAC_{hysail} is a 2-UHF. A corollary of Theorem 1 is that the probability of an attacker forging a check code is negligible. ∎

B. AVAILABILITY PROOF

The next step is related to retrievability and availability and regards the limit of corrupted checks blocks. To achieve such requirement, it is assumed that the adversary $\mathcal{A}$ is capable of accessing the Encoding (VII-A) and Auditing (VII-D) mechanisms. And as explained in V-B, the adversary $\mathcal{A}$ must corrupt more than $(1-\delta)(1+k\delta)n$ blocks to make the original file irretrievable.

As already indicated and will be further detailed in VII-A, check blocks are built as an instance of Online Codes. It is demonstrated in [6] that for given corruption rates used in this approach, with parameters k, δ, n, the probability of successfully retrieving file F is given by $(1 - \delta^k)$.

Theorem 2: The probability of failing to recover the original file is negligible.

Proof: Consider that $|t|$ is total amount of check blocks generated in the Encoding phase and "α" is a fraction of check blocks that may be corrupted. The condition for successful retrievability is that the number of non corrupted blocks, $(1 - \alpha)|t|$ should be equal or greater than the minimum amount necessary to recover the original file F. So, the probability for this requirement is:

$$\Pr[(1 - \alpha)|t| \geq (1 - \delta)(1 + k\delta)n] = 1 - \delta^k \quad (8)$$

Therefore, according to equation (8), it is clear to assume that the probability to successfully recover the original file goes to 1, as δ^k, the probability of failure, is smaller than 2^{-40}. ∎

With Theorems 1 and 2 proved, it is possible to affirm that *Hy-SAIL* is secure in terms of availability and integrity.

Theorem 3: Hy-SAIL is secure regarding integrity and availability of remotely stored data.

Proof: As shown in equation (7), the probability of forging a MAC_{hysail} is equal to $1/2^\lambda$. And as show in the equation (8), the probability to fail in recoverability is approximately to δ^k. Assuming that the modes of failure are independent, the overall probability of failure is $Pr[Fail] = \Pr[Fail_{integrity}] + \Pr[Fail_{recoverability}]$. So the probability of overall success is:

$$1 - Pr[Fail] = 1 - (2^{-\lambda} + \delta^k) \quad (9)$$

which asymptotically is very close to 1 as $2^{-\lambda}$ and δ^k have negligible values. ∎

VII. Hy-SAIL IN DETAILS

In this section, a detailed description of *Hy-SAIL* is presented. It consists of five phases: Encoding, Decoding, Distribution, Auditing and Redistribution. Assuming that a user possesses a file F and wishes to store it remotely in N cloud storage providers. So, the following sequence of steps must be taken:

A. ENCODING

At first, the original file is encoded using Online Codes [6], a special case of Fountain Codes [22], and then the MAC_{hysail} of each message block is computed:

- File F is divided into n message blocks $\{m_i\}_{i=1}^{n}$. If necessary, some padding is added to m_n to keep all of them with the same size in bits.
- In order to add redundancy, an erasure coding is applied over F. This process results in a composite file $F' = \{m'_i\}_{i=1}^{n'}$, with a total of n' blocks equals to $(1+k\delta)n$. The $nk\delta$ blocks added in this initial phase are called auxiliary blocks, which are bitwise XOR of some number of the original message blocks.

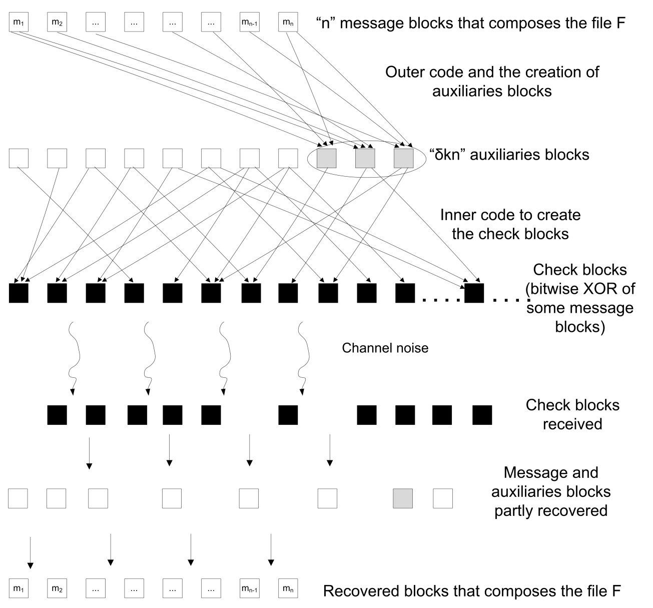
"n" message blocks that composes the file F
Outer code and the creation of auxiliaries blocks
"δkn" auxiliaries blocks
Inner code to create the check blocks
Check blocks (bitwise XOR of some message blocks)
Channel noise
Check blocks received
Message and auxiliaries blocks partly recovered
Recovered blocks that composes the file F

FIGURE 2. **Overview of *Online Codes* adopted at *Hy-SAIL*.**

- The composite file is then supplied to the inner encoding process. Consequently, a stream of check blocks $\{c_i\}_{i=1}^{t}$ is generated, where each check block is the bitwise XOR of a certain number of blocks from the composite file. This process is depicted in figure 2.
- Finally, the user stores a data structure with information about the check blocks c_i and the message blocks labels that generated each of them. The data structure has the form $\langle i, x_i \rangle$, where "i" is index of the check block and "x_i" are blocks labels that generated the check blocks.

As an example of the encoding process, if the 8-th check block is given by $c_8 = m_1 + m_5 + m_{11} + m_{128}$ the data structure has the form $\langle 8, \{1, 5, 11, 128\} \rangle$.

"δ" is the degree of suboptimality and its use in this phase is significant when designing the system. Its value should take into account a number of factors:

1) Corruption frequency of check blocks stored in the cloud storage providers.
2) Storage capacity available in cloud storage providers.
3) Time spent to perform the encoding process.

Algorithm VII-B presents MAC_{hysail}, a hash function algorithm specially suited for the purposes. Assume that all k-binary blocks are represented by polynomials belonging to $\mathbb{F}_{2^k}[X]$ and let FindPrimitive be a routine that receives as parameters an integer k, and returns a primitive polynomial belonging to $\mathbb{F}_{2^k}[X]$. The *MAC* algorithm MAC_{hysail} is to be applied over a file F.

B. DISTRIBUTION

Unlike HAIL [1], *Hy-SAIL* does not make any distinction between primary servers and secondary servers. It only demands basic storage and connectivity requirements in each provider. In other words, *Hy-SAIL* supports a cloud environment with heterogeneous providers.

Algorithm 1

```
Require: A file $F = \{m_i\}_{i=1}^{n} = \{m_1(x)||\dots||m_n(x)\}$ A list of primitive polynomials $\{P_1(x), P_2(x), \dots, P_l(x)\}$.
Ensure: A list of MAC's $MAC_{hysail}(F) = \{h_{i,j}\}_{i=1,j=1}^{n,l} = \{h_{1,1}(x), h_{i,j}(x)||\dots||h_{n,l}(x)\}$
  for $j = 1$ to $l$ do
    $P_j(x) \longleftarrow$ FindPrimitive($\mathbb{F}_{2^k}[X]$).
  end for
  for $i = 1$ to $n$ do
    for $j = 1$ to $l$ do
      $h_{i,j}(x) \longleftarrow m_i(x) \mod P_j(x)$
    end for
  end for
  return $\{h_{1,1}(x)||\dots||h_{i,j}(x)||\dots||h_{n,l}(x)\}$
```

Hy-SAIL is suitable for implementing of *P2P* systems that, even when exposed to high resources of volatility, allows the successful reconstruction of stored files.

As previously mentioned, in the Distribution phase, check blocks are randomly sent to cloud providers ($\mathcal{S}_i$). The distribution must guarantee that check blocks of degree 1 be the first to be recovered in the Decoding phase. The idea is to maintain check blocks in providers that supply a level of retrievability with viable response times.

Assume that the user has a table containing the index of all check blocks and is still in possession of composite file "F'". The following steps are taken to distribute all check blocks across all N cloud storage providers:

- With the MAC_{hysail} of each message block already computed, the user starts the distribution of check blocks.
- In this process, the user selects a number $|t_i|$ of check blocks and sends to a random cloud provider $\mathcal{S}_i$ already authenticated in a private channel.
- After sending all check blocks, the user stores all *MAC* values $\{h_{1,1}(x), \dots, h_{n,l}(x)\}$ and discards all check blocks $\{c_i\}_{i=1}^{t}$ generated in the encoding phase.
- And finally, the user stores a data structure with information about the data that was distributed among cloud storage providers. The data structure has the following form $\langle \mathcal{S}_i, c_i \rangle$, where $\mathcal{S}_i$ is a server identifier and c_i are check blocks labels distributed to server $\mathcal{S}_i$

C. DECODING

In this phase, the goal is to reconstruct the original file "F" with at least $(1 - \delta)(1 + k\delta)n$ random check blocks. First, *Hy-SAIL* performs the search for check blocks with degree 1. These check blocks are mere copies of an original message block m_i. This process is executed through one simple step: find the check blocks that have all message blocks already recovered, except for one. The missing message block is recovered by solving a simple bitwise XOR operation.

For example, suppose that $c_{12} = m_1 \oplus m_5 \oplus m_9 \oplus m_{13}$, such that m_5, m_9 and m_{13} are already known. Thus, $m_1 = c_{12} \oplus m_5 \oplus m_9 \oplus m_{13}$. The process is iteratively repeated until all n message blocks, that compose the original file F, are recovered.

D. AUDITING

Hy-SAIL allows two models, bounded and unbounded, to validate the integrity of check blocks. The difference between them consists in the amount of information collected at the encoding phase and the traffic load needed to perform the auditing phase.

The challenge sent by user depends on the model used in the proof of retrievability scheme. Suppose the user chooses a bounded model in the number of queries, the user sends to the cloud storage provider the following component: a random polynomial $P_l(x)$ from a set of l previously generated and the index of check blocks that will be verified.

In this phase, the cloud storage provider must guarantee the integrity of, at least, $(1-\delta)$ random blocks of the $(1+k\delta n)$ composite messages to retrieve original file F. After each round, a new threshold ($\tau_{provider}$) is calculated which guarantees the retrievability of the file with overwhelming probability. Algorithm VII-D follows a simple example how the bounded model may be applied.

E. REDISTRIBUTION

Unlike other models, *Hy-SAIL* keeps two thresholds that are verified on each round. The first one is related to the internal threshold of each cloud provider ($\tau_{provider}$) that limits the quantity of check blocks that can be corrupted on each epoch. The other is the global threshold (τ_{global}) that limits the total number of check blocks that can be corrupted. This value can be calculated in the *Encoding* phase with the parameters of the error correcting codes.

In this phase, after a verification of integrity, the user can require a new distribution of blocks. In the approach adopted in *Hy-SAIL*, it is not necessary to *download* the entire file to restore corrupted blocks. In order to achieve this, it is performed the bitwise XOR operation in the remaining check blocks that has message blocks which compound the corrupted block. After that, a new one is generated.

Figure 3 illustrates the reconstruction and redistribution of encoded blocks:

- **Step 1**: If any problem occurs with the cloud storage provider, the Redistribution mechanism is activated.
- **Step 2**: Upon receiving information about failure in the integrity check of a given check block, the bitwise XOR is performed with the content of two or more check blocks to reconstruct the corrupted block. **(For instance:** $c_{j+1} = c_{j-6} \oplus c_{j-1}$**).**
- **Step 3**: The recently obtained check block (c_{j+1}) is sent to another cloud storage provider.

Algorithm 2

Require:

- A list of *MAC's*, which are represented by: $MAC_{hysail}(x) = \{h_{1,1}(x), \ldots, h_{i,j}(x)\}$.
- A list of check blocks c_t's encoded in the Encoding Phase.
- A list of primitive polynomials $P_l(x)$ generated in the Encoding Phase.
- Thresholds $\tau_{provider}$ and $\tau_{maximum}$.

Ensure:

- Verification of the existence of corrupted blocks.

```
User
1 - The client chooses a random and primitive polynomial $P_j(x)$.
2 - The client chooses a random subset of $c_t$'s:
$\Longrightarrow list \longleftarrow c_\eta$
3 - The client sends a challenge $list$, $P_j(x)$ to a random cloud provider $\mathcal{S}_i$.

Cloud Storage Provider
1 - $A(x) \longleftarrow (\bigoplus_{\eta \in list} c_\eta(x)) \bmod P_j(x)$.
2 - Send the response $A(x)$ to the User.

User
if $\left(\bigoplus_{i \in list} c_\eta(x) \bmod P_j(x)\right) = A(x)$ then
    The level of integrity is acceptable:
    $\tau_{provider} \geq \tau_{maximum}$
else
    The client is left with two options:
    1 - Perform the process of verification of integrity in another cloud storage provider.
    2 - Initiate the Redistribution phase.
end if
```

VIII. IMPLEMENTATION RESULTS

Hy-SAIL was implemented, as a proof of concept, in a non optimized fashion in *Python* programming language, version 2.7.3, with different parameters for encoding a file F. The experiments were conducted in a machine with the following settings: MacBook Pro, Intel Core i7 quad core, 2,3GHz with 6 MB cache L3, RAM memory 8 GB, Hard Disk 500 GB with 5400 RPM.

Check blocks were generated according to the probability distribution shown in [6]. Each check block is obtained by performing the bitwise XOR for a set of t randomly chosen message blocks,. The value t is known as the check block's degree and is chosen according to a probability distribution

TABLE 1. Comparison of PoR Models: "n" is the size of message and "k" is the safety parameter.

Itens to be compared in PoR models	Juels-Kaliski [7]	Shacham-Waters [8]	HAIL [1]	Hy-SAIL
Encoding complexity of the file to be stored in the remote provider	$\mathcal{O}(k(n-k)\log_2 n)$	$\mathcal{O}(k(n-k)\log_2 n)$	$\mathcal{O}(k(n-k)\log_2 n)$	$\mathcal{O}$(n)
Decoding complexity of the file to be stored in the remote provider	$\mathcal{O}(k(n-k)\log_2 n)$	$\mathcal{O}(k(n-k)\log_2 n)$	$\mathcal{O}(k(n-k)\log_2 n)$	$\mathcal{O}$(1)
Storage complexity in the provider	$\mathcal{O}$(2n)	$\mathcal{O}$(2n)	$\mathcal{O}$(n)	$\mathcal{O}$(n)
Possibility of Multi-Provider Storage	NO	NO	YES	YES
Access to the blocks of the files in the realization of the PoR	PARTIAL	PARTIAL	TOTAL	TOTAL
Redistribution of corrupted blocks	NO	NO	NO	YES
Unlimited number of times for blocks integrity query	NO	YES	NO	YES

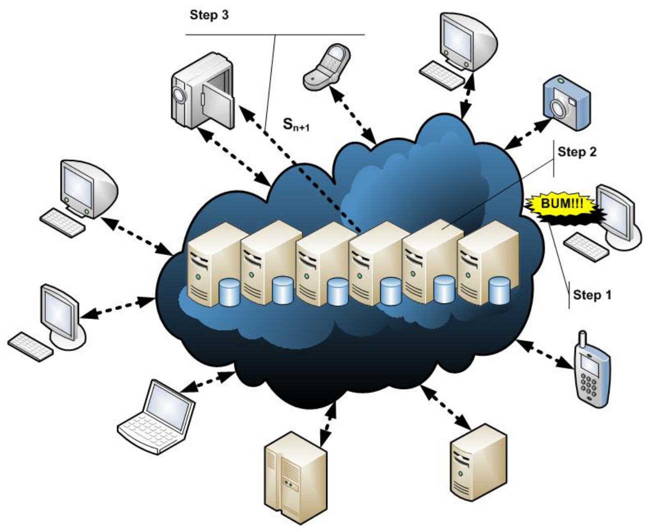

FIGURE 3. Redistribution of a new block.

$\rho = (\rho_1, \ldots, \rho_t \ldots, \rho_D)$, where ρ_t denotes the probability of having a check block with degree t, and D is a constant denoting the maximum degree of a check block. For any given ϵ and δ, the probability distribution ρ is defined as follows:

$$\rho_1 = 1 - \frac{1 + 1/D}{1 + \varepsilon} \quad \text{and} \quad \rho_i = \frac{(1 - \rho_1)}{(1 - 1/D)\, i\, (i-1)}, \tag{10}$$

where the maximum degree is given by:

$$D = \left\lceil \frac{\ln(\varepsilon/2) + \ln(\delta)}{\ln(1 - \delta)} \right\rceil. \tag{11}$$

Figure 4 shows the probability of each degree in the set of check blocks. "Function 1" has parameters $\delta = 0,005$ and $\epsilon = 0,01$, and "Function 2" has parameters $\delta = 0,01$ and $\epsilon = 0,02$. In spite of the different parameters, both functions have similar probability distributions in the construction of check blocks.

Another relevant aspect is the value of n, which has direct implications in the size and in the quantity of message blocks m_i. The size of the file F and the parameters chosen in the Encoding phase (δ,ϵ,k,n) reflects the computational effort required to generate check blocks. The main goal is to show how the time behavior in according to value n.

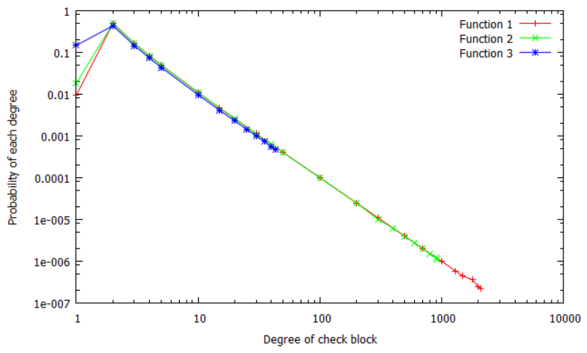

FIGURE 4. Probability X Degree of the encoded block.

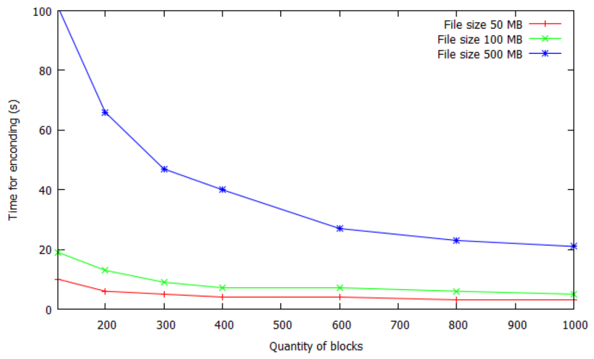

FIGURE 5. Encoding time.

With parameters $\delta = 0,05$ and $\epsilon = 0,10$, figures 5 and 6 show the time spent encoding and decoding files with sizes 50 MB, 100 MB, 500 MB for different values of n.

A concern regarding error correcting codes is the probability of retrieving the original file. *Hy-SAIL* allows the use of a random number of check blocks to reconstruct the file. Figure 7 shows the relation between the probability of failure and the number of check blocks. It is used as parameters: $\delta = \epsilon/2$, a fixed $k = 3$ and $n = 500$. The probability of failure in the recovery process is given by δ^k, and the amount of check blocks necessary to recover the original file is equal to $(1 - \delta)(1 + k\delta)n$.

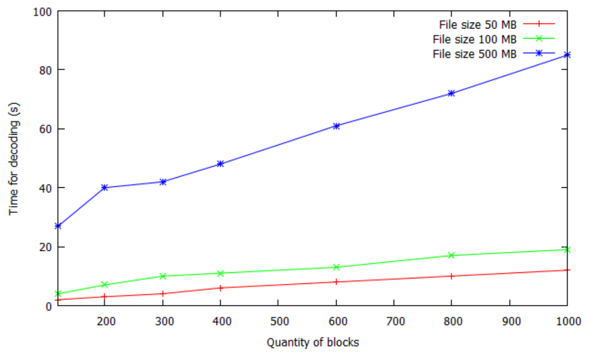

FIGURE 6. **Decoding time.**

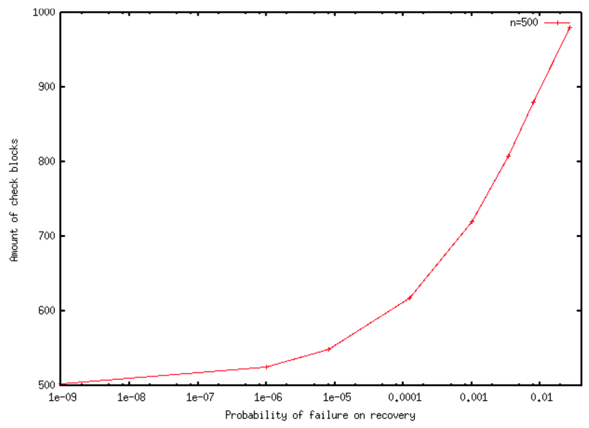

FIGURE 7. **Number of check blocks X Probability of failure on recovery.**

IX. CONCLUSION

Hy-SAIL provides a high scalability, availability and integrity layer which addresses the most common needs of data storage in a cloud. *Hy-SAIL* has an abstraction layer implemented through error correcting codes, ensuring availability against a realistic adversarial model and retrievability of files stored remotely even if a fraction is corrupted. It also implemented an integrity check protocol that provides security and scalability for *PoR* schemes and improves over previous proposals concerning processing, storage and bandwidth complexity.

Hy-SAIL has interesting applications to explore, as a peer-to-peer implementation which fits in its architecture, having no distinction among storage providers. With its flexibility, it is possible to construct two models for integrity checks: Bounded and Unbounded. The difference between them is the amount of information kept with the user and the traffic generated to perform block integrity ''challenge-response'' checks. Compared to other schemes, *Hy-SAIL* still needs a model for public verifiability, delegating to a trusted third party periodical integrity checks.

With the results collected by the unoptimized proof of concept implementation, it is clear that *Hy-SAIL* is viable and practical.

Table 1 presents a comparison of *Hy-SAIL* with related systems. It indicates that *Hy-SAIL* improves several aspects like: less complexity to coding and decoding the file to store remotely, the ability to redistribute the portions of file and the possibility to check the integrity an unlimited number of times.

Moreover, it offers novel contributions for file systems in distributed environments and may serve as a basis for aggregating other security properties, like confidentiality, on top of integrity and availability, while preserving the approach's scalability characteristics.

ACKNOWLEDGMENT

The authors also thank Anderson Nascimento for insightful remarks in early stages of this work, and Josias Gonzaga for his contribution implementing the PoC version of *Hy-SAIL*.

REFERENCES

[1] K. D. Bowers, A. Juels, and A. Oprea, ''HAIL: A High-availability and integrity layer for cloud storage,'' in *Proc. 16th ACM Conf. Comput. Commun. Secur.*, New York, NY, USA, 2009, pp. 187–198.

[2] N. Carr, *The Big Switch—Rewiring the World, From Edison to Google*. New York, NY, USA: Norton, 2009.

[3] (2011). *Amazon's Cloud Crash Disaster Permanently Destroyed Many Customers, Data*. Accessed: Dec. 16, 2019. [Online]. Available: http://www.businessinsider.com/amazon-lost-data-2011-4

[4] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, ''Provable data possession at untrusted stores,'' in *Proc. 14th ACM Conf. Comput. Commun. Secur.*, New York, NY, USA, 2007, pp. 598–609.

[5] S. Halevi, D. Harnik, B. Pinkas, and A. Shulman-Peleg, ''Proofs of ownership in remote storage systems,'' in *Proc. 18th ACM Conf. Comput. Commun. Secur. (CCS)*, New York, NY, USA, 2011, pp. 491–500.

[6] P. Maymounkov, ''Online codes,'' New York Univ., New York, NY, USA, Tech. Rep. TR2002-833, 2002.

[7] A. Juels and B. S. Kaliski, Jr., ''PORs: Proofs of retrievability for large files,'' in *Proc. 14th ACM Conf. Comput. Commun. Secur. (CCS)*, 2007, pp. 584–597.

[8] H. Shacham and B. Waters, ''Compact proofs of retrievability,'' in *Proc. AsiaCrypt*, Springer-Verlag, 2008, pp. 90–107.

[9] Y. Dodis, S. Vadhan, and D. Wichs, ''Proofs of retrievability via hardness amplification,'' in *Proc. 6th Theory Cryptogr. Conf. Theory Cryptogr. (TCC)*. San Francisco, CA, USA, Springer-Verlag, 2008, pp. 109–127. Accessed: Dec. 16, 2018. doi: 10.1007/978-3-642-00457-5_8.

[10] K. D. Bowers, A. Juels, and A. Oprea, ''Proofs of retrievability: Theory and implementation,'' in *Proc. ACM Workshop Cloud Comput. Secur. (CCSW)*, New York, NY, USA, 2009, pp. 43–54. Accessed: Dec. 16, 2018. doi: 10.1145/1655008.1655015.

[11] Y. Zhu, H. Wang, Z. Hu, G.-J. Ahn, H. Hu, and S. S. Yau, ''Efficient provable data possession for hybrid clouds,'' in *Proc. 17th ACM Conf. Comput. Commun. Secur. (CCS)*, New York, NY, USA, 2010, pp. 756–758. doi: 10.1145/1866307.1866421.

[12] C. Wang, Q. Wang, K. Ren, and W. Lou, ''Privacy-preserving public auditing for data storage security in cloud computing,'' in *Proc. 29th Conf. Inf. Commun. (INFOCOM)*, Piscataway, NJ, USA: IEEE Press, 2010, pp. 525–533. [Online]. Available: http://portal.acm.org/citation.cfm?id=1833515.1833620

[13] Q. Wang, C. Wang, J. Li, K. Ren, and W. Lou, ''Enabling public verifiability and data dynamics for storage security in cloud computing,'' in *Proc. 14th Eur. Conf. Res. Comput. Secur. (ESORICS)*, Berlin, Germany: Springer-Verlag, 2009, pp. 355–370. [Online]. Available: http://dl.acm.org/citation.cfm?id=1813084.1813114

[14] C. Wang, Q. Wang, K. Ren, and W. Lou, ''Ensuring data storage security in cloud computing,'' in *Proc. 17th IEEE Int. Workshop Qual. Service (IWQoS)*, 2009.

[15] D. Boneh, B. Lynn, and H. Shacham, ''Short signatures from the weil pairing,'' in *Proc. 7th Int. Conf. Theory Appl. Cryptol. Inf. Secur., Adv. Cryptol. (ASIACRYPT)*. London, U.K.: Springer-Verlag, 2001, pp. 514–532. [Online]. Available: http://portal.acm.org/citation.cfm?id=647097.717005

[16] M. Bellare and P. Rogaway, ''Random oracles are practical: A paradigm for designing efficient protocols,'' in *Proc. 1st ACM Conf. Comput. Commun. Secur. (CCS)*, New York, NY, USA, 1993, pp. 62–73. doi: 10.1145/168588.168596.

[17] A. J. Menezes, P. C. Van Oorschot, S. A. Vanstone, and R. L. Rivest, *Handbook of Applied Cryptography*. 1997.

[18] P. Maymounkov and D. Mazières, "Rateless codes and big downloads," in *Peer-to-Peer Systems II* (Lecture Notes in Computer Science), vol. 2735, M. F. Kaashoek and I. Stoica, Eds. Berlin, Germany: Springer, 2003, pp. 247–255. [Online]. Available: http://dblp.uni-trier.de/db/conf/iptps/iptps2003.html#MaymounkovM03

[19] J. Guajardo, T. Güneysu, S. S. Kumar, C. Paar, and J. Pelzl, "Efficient hardware implementation of finite fields with applications to cryptography," *Acta Appl. Math.*, vol. 93, nos. 1–3, pp. 75–118, 2006.

[20] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *Proc. IEEE 26th Symp. Mass Storage Syst. Technol. (MSST)*. Washington, DC, USA: IEEE Comput. Soc., 2010, pp. 1–10. doi: 10.1109/MSST.2010.5496972.

[21] A. Juels, J. Kelley, R. Tamassia, and N. Triandopoulos, "Falcon codes: Fast, authenticated LT codes (Or: Making rapid tornadoes unstoppable)," in *Proc. 9th Annu. ACM Symp. Theory Comput. (STOC)*, New York, NY, USA, 1977, pp. 106–112. doi: 10.1145/800105.803400.

[22] D. J. C. Mackay, "Fountain codes," *IEEE Commun.*, vol. 152, no. 6, pp. 1062–1068, Dec. 2005.

[23] A. Juels, J. Kelley, R. Tamassia, and N. Triandopoulos, "Falcon codes: Fast, authenticated LT codes," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.*, Denver, CO, USA, 2015, pp. 1032–1047.

[24] M. M. Etemad and A. Küpçü, "Generic efficient dynamic proofs of retrievability," in *Proc. ACM Cloud Comput. Secur. Workshop (CCSW)*, Vienna, Austria, 2016, pp. 85–96.

[25] M. B. Paterson, D. R. Stinson, and J. Upadhyay, "Multi-prover proof of retrievability," *J. Math. Cryptol.*, vol. 12, no. 4, pp. 203–220, 2018.

[26] M. B. Paterson, D. R. Stinson, and J. Upadhyay, "A coding theory foundation for the analysis of general unconditionally secure proof-of-retrievability schemes for cloud storage," *J. Math. Cryptol.*, vol. 7, no. 3, pp. 183–216, 2013.

[27] B. Wang and X. Hong, "Multi-file proofs of retrievability for cloud storage auditing," Cryptol. ePrint Arch., Tech. Rep. 2013/348, 2013.

DINO MACEDO AMARAL received the degree in computer science and the M.Sc. and Ph.D. degrees in electrical engineering from the University of Brasília (UnB), in 2003, 2009, and 2013, respectively. Since 2002, he has been a Senior Analyst with the Bank of Brazil, where he created a protocol for two-factor authentication (2FA), with a patent registered under Brazilian laws at INPI, with 2 million active users without any incident of fraud. His research interests include distributed systems, big data, real-time systems, and machine learning.

JOÃO J. C. GONDIM received the degree in electrical engineering (electronics) from the Federal University of Pernambuco (UFPE), Recife, Brazil, in 1984, the M.Sc. degree in computing science with the Imperial College, University of London, in 1987, and the Ph.D. degree in electrical engineering from the University of Brasília (UnB), in 2017. Since 1994, he has been a Lecturer with the Department of Computing Science (CIC), UnB where he is currently a tenured member of faculty. His research interests include information and cybersecurity.

ROBSON DE OLIVEIRA ALBUQUERQUE graduated in computer science from the Catholic University of Brasília, in 1999. He received the MBA degree in computer networks from the Educational Union of Brasília, in 2001, the master's degree in electrical engineering from the University of Brasília, in 2003, the D.E.A. degree from the University Complutense of Madrid, in 2007, the Ph.D. degree in electrical engineering from the University of Brasília, Brazil, in 2008, and the Ph.D. degree in information systems from the University Complutense of Madrid, Spain, in 2016. He is currently a Researcher with the University of Brasília and a member of the Group of Analysis, Security and Systems research group, University Complutense of Madrid. His research interests include cybersecurity, network security, information security, distributed systems, and computer networks.

ANA LUCILA SANDOVAL OROZCO was born in Chivolo, Magdalena, Colombia, in 1976. She received the Engineering degree in computer science from the Universidad Autónoma del Caribe, Colombia, in 2001, and the M.Sc. degree in research in computer science and the Ph.D. degree in computer science from the Universidad Complutense de Madrid, Spain, in 2009 and 2014, respectively. She holds a Specialization Course in computer networks from the Universidad del Norte, Colombia, in 2006. She is currently a Postdoctoral Researcher with the Universidad Complutense de Madrid. Her main research interests include coding theory, information security, and its applications.

LUIS JAVIER GARCÍA VILLALBA received the degree in telecommunication engineering from the Universidad de Málaga, Spain, in 1993, and the M.Sc. degree in computer networks and the Ph.D. degree in computer science from the Universidad Politécnica de Madrid, Spain, in 1996 and 1999, respectively. He was a Visiting Scholar with the Computer Security and Industrial Cryptography (COSIC) Group, Department of Electrical Engineering, Faculty of Engineering, Katholieke Universiteit Leuven, Belgium, in 2000, and a Visiting Scientist with the IBM Research Division, IBM Almaden Research Center, San Jose, CA, USA, from 2001 to 2002. He is currently an Associate Professor with the Department of Software Engineering and Artificial Intelligence, Universidad Complutense de Madrid (UCM), and the Head of the Group of Analysis, Security and Systems (GASS) research group, School of Computer Science, UCM. His professional experience includes research projects with Hitachi, IBM, Nokia, and Safelayer Secure Communications.