INTERNATIONAL SYMPOSIUM ON NETWORK CODING YEAR 2012

Auditing for Network Coding Storage

Anh Le, *Student Member, IEEE,* and Athina Markopoulou, *Member, IEEE*

Abstract—Network coding-based storage has recently received a lot of attention in the network coding community. Independently, another body of work has proposed integrity checking schemes for cloud storage, none of which, however, is customized for network coding storage or can efficiently support repair. In this work, we bridge the gap between these currently disconnected bodies of work, and we focus on the (novel) advantage of network coding for integrity checking. We propose NC-Audit – a remote data integrity checking scheme, designed specifically for network coding-based storage cloud. NC-Audit provides a unique combination of desired properties: (i) efficient checking of data integrity (ii) efficient support for repairing failed nodes (iii) full support for modification of outsourced data and (iv) protection against information leakage when checking is performed by a third party. The key ingredient of the design of NC-Audit is a novel combination of SpaceMac, a homomorphic MAC scheme for network coding, and NCrypt, a novel CPA-secure encryption scheme that is compatible with SpaceMac. Our evaluation of a Java implementation of NC-Audit shows that an audit costs the storage node and the auditor only a few milliseconds of computation time, and lower bandwidth than prior work.

1 Introduction

Fundamental to cloud computing is the ability to store user data reliably on the storage cloud. If the original data consists of K packets, an (N, K) maximum distance separable (MDS) code is typically used to produce N packets to be stored individually on N storage nodes, thus tolerating up to $(N - K)$ node failures. Network coding (NC) has been shown to achieve the minimum *repair bandwidth* – much less than K packets, which is required to reconstruct the original data [1], [2]. The key ingredients of NC-based distributed storage include (i) *subpaketization*, *i.e.*, each storage node stores *subpackets* (or blocks) that are linear combinations of blocks that form the original data, and (ii) *subpacket mixing* when repairing. An example is given in Fig. 1. However, repair bandwidth is only one aspect of cloud storage.

Another practical aspect, which has received only modest attention in the network coding community, is integrity checking of the data stored on the cloud. Data can be lost or corrupted for various reasons without the user being aware of it. For example, storage errors, such as torn writes [3] and latent errors [4], may damage the data in a way that is not detected. Data storage providers also have incentives to cheat: *e.g.*, some providers do not report data loss incidents in order to maintain their reputation [5]–[7]. This problem is further exacerbated in NC-based systems because corrupted data on one storage node can propagate to many other nodes during the repair process. Therefore, it is important for the user to be able to audit the integrity of the data stored on the cloud.

However, considering a large file stored on the cloud, the ability to audit this file regularly may be out of the ability or budget of users with limited resources [7], [8]. Therefore, users often resort to a third party to perform the audit on their behalf [5], [7], [9], [10]. In this latter case, it is important that the auditing protocol be privacy-preserving, *i.e.*, it should not leak the data to the third party [7], [11]. Indeed, the users can leverage data encryption to protect their data before outsourcing it [10]. However, data encryption should be complementary and orthogonal to integrity checking protocols. Furthermore, the users may want to outsource unencrypted instead of encrypted data to support more efficient and/or complex computation over the data.

Although there is a rich literature on auditing protocols for cloud storage in general [5]–[7], [9]–[17], there have been very few auditing protocols for NC-based distributed storage systems [18], [19]. These protocols, however, are generic in the sense that they do not specifically exploit network coding properties for efficient integrity checking [18]. Furthermore, they do not efficiently support repair or data dynamics [18], and do

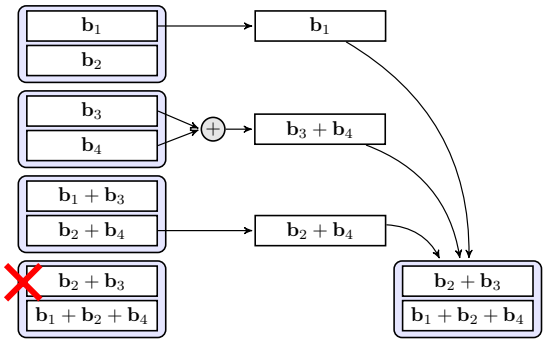

Fig. 1. Repairing a failed node [1]: The original data consists of four blocks: $\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3$ and $\mathbf{b}_4$. A $(4, 2)$ MDS code is used such that any 2 nodes can be used to restore the original data. Note that the repair involves combining blocks $\mathbf{b}_3$ and $\mathbf{b}_4$ and the repair bandwidth consists of 3 blocks instead of 4, which is needed to reconstruct the whole data.

- *A. Le (anh.le@uci.edu) is with the Computer Science Department and A. Markopoulou (athina@uci.edu) is with the Electrical Engineering and Computer Science Department, UC Irvine, USA.*

not prevent data leakage [18], [19].

In this work, we propose a symmetric key-based cryptographic protocol, called NC-Audit, to check for the integrity of data stored on a NC-based distributed storage system. To the best of our knowledge, this is the first scheme proposed for NC-based systems that possesses all the following properties:

(i) **Efficient Integrity Checking**: The integrity check incurs a small bandwidth and computational overhead (few milliseconds). It guarantees that, with high probability, the storage provider passes the integrity check if and only if it possesses the data. The proposed protocol also supports unlimited number of checks.

(ii) **Efficient Support for Repair and Data Dynamics**: The repair of failed nodes and the changes made to the data (including update, append, insert, and delete operations) require negligible bandwidth (no data download) and computation (sub milliseconds) for maintaining the metadata used by the integrity checking.

(iii) **Efficient Privacy Protection**: A third party auditor cannot learn any information about the user data through the checking protocol, except for the metadata used by the integrity checking. This privacy preserving property incurs a small bandwidth (0.4%) and computational overhead (few milliseconds).

We would like to emphasize that, independently of (iii), properties (i) and (ii) together are already useful and of interest to users who prefer to audit the data themselves; furthermore, NC-Audit is also the first protocol that possesses (i) and (ii) at the same time. In addition, NC-Audit is the first auditing scheme that fully exploits network coding by design. The key ingredient of NC-Audit is a novel combination of SpaceMac – a homomorphic authenticator that was previously specifically designed for network coding [20], [21], and NCrypt – a novel encryption scheme that exploits random linear combinations so as to be compatible with SpaceMac (Section 4.4).

We implemented NC-Audit in Java, utilizing our previous implementation of SpaceMac [21]. Our evaluation of NC-Audit shows that it has very low computational overhead: when performing an audit, both the storage node and the third party auditor only needs to spend a couple of milliseconds.

The rest of the paper is organized as follows. In Section 2, we discuss related work. In Section 3, we formulate the problem and describe the threat model. In Section 4, we describe the auditing framework and the key building blocks of NC-Audit (SpaceMac and NCrypt) before presenting NC-Audit itself. In Section 5, we show how NC-Audit efficiently supports repair and data dynamics. In Section 6, we analyze the security of NC-Audit. In Section 7, we evaluate its storage, bandwidth, and computational efficiency. In Section 8, we conclude.

2 Related Work

2.1 Integrity Checking for Remote Data

There has been a rich body of work on integrity checking for remote data [5]–[7], [9]–[17], known as *Proof of Retrievability* and *Proof of Data Possession*.

Proof of Retrievability (POR). In [10], Juels and Kaliski introduced the notion of POR, where a POR enables a client (verifier) to determine that the server (prover) possesses a file or data object. Furthermore, a successful execution of POR would allow a verifier to extract the file from the proof. The main POR scheme presented in this work uses *sentinels*, *i.e.*, small check blocks, that are inserted into the outsourced data to guard against large file corruption. At the same time, it also utilizes error correcting codes to protect against small file corruption. This scheme can only handle a limited number of queries, which has to be fixed a priori. NC-Audit does not use sentinels and supports unlimited number of queries.

In [9], Shacham and Waters proposed two POR schemes with full proofs of security and extract-ability. The first one, built on BLS signatures, provides public verifiability. The second one, built on pseudorandom functions (PRFs), provides private verifiability. Both of these schemes exploit homomorphic properties to aggregate authenticator values. NC-Audit also exploits homomorphic properties and provides private verifiability.

Proof of Data Possession (PDP). The notion of PDP was introduced by Ateniese *et al.* [5]. The PDP scheme in [5] uses homomorphic RSA signatures to generate verification tags. The data possession guarantee provided by this scheme is under the RSA and KEA1 [22] assumptions in the random oracle model. As discussed in [9], the notion of PDP is considered weaker than POR. This is because in POR, a successful audit guarantees that all the data can be extracted while in PDP, only a certain percentage of the data (*e.g.*, 90%) is guaranteed to be available. We will show that NC-Audit provides the stronger data possession as in POR (Section 6.1).

Data Dynamics. In [12], Ateniese *et al.* proposed a symmetric-key based checking scheme that supports data dynamics. This scheme is built on regular PRFs, hash functions, and encryptions. It provides private verifiability and only supports a limited number of queries. In [14], Erway *et al.* proposed an auditing scheme built on rank-based authenticated skip lists and requires the storage server to maintain the lists for verification. It provides private verifiability but could be extended to provide public verifiability [14]. In [15], Wang *et al.* proposed a public auditing scheme that uses a combination of the BLS-based scheme in [9] and Merkle Hash Tree (MHT). NC-Audit supports data dynamics and does not require data block download (blockless) in all operations (Section 5.2). The approach taken by NC-Audit is similar to [12] but different from [15] and [14], where the changes are immediately verified by the user.

Privacy Preserving. In [6], Shah *et al.* proposed an auditing protocol that is privacy preserving. This protocol first encrypts the data and then send a number of message authentication code (MAC) tags of the encrypted data to the auditor. The auditor verifies both the outsourced data and the outsourced encryption key. This approach only works on encrypted files. It also requires the auditor to maintain states and supports only limited number of audits. In [11], Wang *et al.* also proposed a privacy preserving auditing protocol that has public verifiability. This protocol can be considered an extension of the BLS-based protocol in [9]. In this approach, the aggregated (proving) block sent by the storage server is masked with a random element to protect the privacy of the block. NC-Audit is explicitly designed to provide privacy preserving-auditing (Section 4.5 and 6.2). In particular, NC-Audit provides privacy by encrypting the response block.

We stress that none of the schemes described above was customized for NC-based storage; thus, they do not provide efficient support for node repair. NC-Audit was designed to achieve all the above good properties while efficiently supporting repair.

2.2 Integrity Checking for NC-based Storage Systems

NC-based Storage Systems. The benefits of network coding for distributed storage were first formalized by the work of Dimakis *et al.* [2]. In particular, in [2], the authors proposed the notion of *regenerating codes* and show that they can significantly reduce the repair bandwidth. This work showed the fundamental tradeoff between node storage and repair bandwidth and proposed regenerating codes that can achieve any point on the optimal tradeoff curve. An excellent survey on recent advances in NC-based storage system can be found at [1]. A wiki on NC-based storage cloud is maintained at [23]. NC-Audit is designed to fully support regenerating codes.

One of the first implementations of NC-based storage cloud is NCCloud by Hu *et al.* [24]. In particular, NCCloud is a proxy-based system for *multiple-cloud* storage. It utilizes a functional minimum-storage regenerating code to provide cost-effective repair for a permanent single-cloud failure. This efficient repair is achieved without the cost of storage or redundancy level. NCCloud prototype was deployed atop Windows Azure Storage.

Integrity Checking Schemes. There have been only a few work that provide remote data checking for NC-based storage. In [19], Dikialotis *et al.* proposed an integrity checking scheme that utilizes the error-correction capabilities of the storage system. This scheme aims to detect errors with a very small amount of bandwidth. The key technique for reducing the bandwidth is to project data blocks onto a small random vector. This checking scheme is inherently different from NC-Audit as it relies on the communication between the auditor and multiple nodes to perform a single check while NC-Audit does not. Moreover, this scheme is information-theory based while NC-Audit leverages cryptographic primitives to provide the checking.

A more recent integrity checking scheme for NC-based storage was proposed in [18]. In this work, Chen *et al.* adopted the symmetric-key based scheme that Shacham and Waters proposed for regular cloud storage [9] with minor modification. In particular, atop of the symmetric-key based scheme in [9], the scheme in [18] proposed to encrypt the coding coefficients of the outsourced encoded blocks to prevent *replay attacks*, where a malicious storage node may store old (incorrect) encoded blocks instead of the new (correct) encoded blocks as required by the repair [18]. NC-Audit overcomes this attack by requiring the user/auditor to store the coding coefficients, which is needed for the repair process and only occupies a small amount of storage and could be made constant (Section 7.1).

The scheme in [18] also protects the repair phase against pollution attacks, *i.e.*, preventing remaining nodes from sending corrupted data to the new (recovering) node. In [18], the authors proposed that the user acts as the middle man, *i.e.*, downloads necessary blocks for repair, checks for their integrity, and constructs and sends the new blocks to the new node. This approach puts a heavy bandwidth and computational overhead on the user. Dealing with pollution attacks is out of the scope of this work. We refer the reader to the rich literature, including our previous work, that deal with pollution attacks [25]–[31]. We stress that when leveraging other pollution defense approaches [25]–[31], the new node can detect a pollution attack itself without resorting to the user acting as the middle man, thereby eliminating both user's bandwidth and computational overhead.

Finally, the scheme in [18] neither supports data dynamics nor privacy-preserving auditing while NC-Audit does. We provide detailed performance comparison between NC-Audit and [18] in Section 7.

Other Security Issues. Other security problems for NC-based storage include protecting the privacy and integrity of the blocks while repairing. On one hand, the work in [32] and [33] prevent eavesdroppers from accessing/decoding all the data. In [32], Pawar *et al.* provide an explicit code construction that achieves the secrecy capacity for the *bandwidth-limited regime* of the storage systems under repair dynamics. In [33], Rouayheb *et al.* analyze the effects of interaction between the storage nodes on the amount of data revealed to the eavesdroppers. On the other hand, the work in [34] and [35] aim to provide protection against pollution attacks during the repair. In [34], the Pawar *et al.* provide upper bounds on the maximum amount of information that can be stored safely when there are malicious nodes. In [35], Buttyan *et al.* provide a lightweight, pollution-resilient decoding

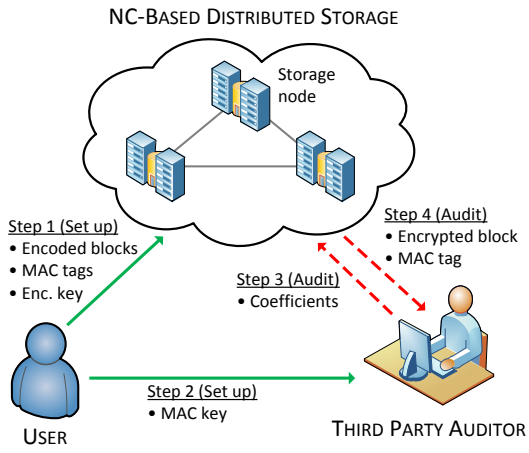

Fig. 2. Parties and Steps Involved in NC-Audit.

algorithm that is capable of finding adversarial blocks.

2.3 Our Work in Perspective

The preliminary version of this work has appeared in [36]. This work is an important extension of our previous 6-page version [36]. In particular, this work provides complete proofs of all theorems. It also provides a complete discussion of all data update operations, including block update, append, insert, and delete. Furthermore, we discuss and compare our storage overhead with the prior work [11], [15], [18]. Finally, we provide this work with a comprehensive discussion of related literature.

3 Problem Formulation

3.1 System Model and Operations

Fig. 2 illustrates an overview of the system. We consider a cloud storage service that involves three entities: a user, NC-based storage nodes, which make up the storage cloud, and a third party auditor (TPA). The user distributes her data on the storage nodes and may also dynamically update her data. The user resorts to a TPA to check for the integrity of her data stored at each node; at the same time, she does not want the TPA to learn about her data. We assume that the user is responsible for maintaining the data stored at each storage node. Our work, however, is also applicable to the scenario where there is a cloud service provider who is independent from the user and is responsible for maintaining the storage cloud.

The user follows the following basic steps to store her data on the storage cloud. We adopt the notations used in [30]. Denote the original file by $\mathcal{F}$. The user first divides $\mathcal{F}$ into m blocks, $\hat{\mathbf{b}}_1, \cdots, \hat{\mathbf{b}}_m$. Each block is a vector in an n-dimensional linear space $\mathbb{F}_q^n$, where $\mathbb{F}$ is a finite field of size q. To facilitate the decoding, the user then augments each block $\hat{\mathbf{b}}_i$ with its m *global coding coefficients*. The resulting blocks, $\mathbf{b}_i$, have the following form:

$$\mathbf{b}_i = (\overbrace{—\hat{\mathbf{b}}_i—}^{n}, \overbrace{\underbrace{0, \cdots, 0, 1}_{i}, 0, \cdots, 0}^{m}) \in \mathbb{F}_q^{n+m}.$$

We call $\mathbf{b}_i$ *source blocks* and the space spanned by them *source space*, denoted by Π. We use $\mathsf{aug}(\mathbf{b_i})$ to denote the coefficients of $\mathbf{b}_i$. Typically, $n \gg m$, and this presentation is also called an n-extended version of a storage code [19].

The user then creates a number of encoded blocks using an appropriate linear coding scheme for the desired reliability, *e.g.*, an array MDS evenodd code is used in Fig. 1. Each encoded block is a linear combination of the source blocks. Note that if an encoded block $\mathbf{e}$ equals $\sum_{i=1}^{m} \alpha_i \mathbf{b}_i$, then the last m coordinates of $\mathbf{e}$ are exactly the coding coefficients α_i's. These encoded blocks are then distributed across the N storage nodes of the storage cloud. Let M be the number of encoded blocks stored at a storage node. In the example given in Fig. 1, $m = 4$, $N = 4$, and $M = 2$.

3.2 Threat Model

We adopt the threat model considered in [11] and [16]. In particular, we consider semi-trusted storage nodes who behave properly and do not deviate from the prescribed protocol. However, for their own benefit, the nodes may deliberately delete rarely accessed user's data. They may also decide to hide data corruptions, caused by either internal or external factors, to maintain reputation. For clarity, we focus our discussion on a single storage node except when discussing the repair process.

Similar to [11], we assume that the TPA, who is in the business of auditing, is reliable and independent. The TPA has no incentive to collude with the user or the storage node during the auditing process. The TPA, however, must not be able to learn any information about the user's data through the auditing process, aside from the metadata needed for the auditing.

In summary, the threat model includes a malicious storage node, who wants to hide data corruption, and a TPA, who wants to learn about the user's data. We assume that both the node and the TPA are fully aware of all the cryptographic constructions and protocols used; however, their runtime is polynomial in the security parameter.

4 Auditing Scheme

4.1 Definitions and Auditing Framework

We follow the literature on checking the integrity of remote data [5], [9]–[11], [13] and adapt the proposed framework to our privacy-preserving auditing system. In particular, we consider an auditing scheme which consists of four algorithms:

- $\mathsf{KeyGen}(1^\lambda) \rightarrow (k_1, k_2)$ is a probabilistic key generation algorithm that is run by the user to setup the scheme. It takes a security parameter λ as input and outputs two different private keys, k_1 and k_2. k_1 is used to generate verification metadata, and k_2 is used to encrypt the possession proof.
- $\mathsf{TagGen}(\mathbf{e}, k_1) \rightarrow t$ is a probabilistic algorithm run by the user to generate the verification metadata. It

takes as input a coded block, $\mathbf{e}$, a private key, k_1, and outputs a verification data of $\mathbf{e}$, t.

- $\mathsf{GenProof}(k_2, (\mathbf{e}_1, \cdots, \mathbf{e}_M), (t_{\mathbf{e}_1}, \cdots, t_{\mathbf{e}_M}), \mathsf{chal}) \rightarrow V$ is run by the storage node to generate a proof of possession. It takes as input a secret key, k_2, coded blocks stored at the node, $\mathbf{e}_1, \cdots, \mathbf{e}_M$, their corresponding verification metadata, $t_{\mathbf{e}_1}, \cdots, t_{\mathbf{e}_M}$, and a challenge, chal. It outputs a proof of possession, V, for the coded blocks determined by chal.
- $\mathsf{VerifyProof}(k_1, \mathsf{chal}, V) \rightarrow \{1, 0\}$ is run by the user in order to validate a proof of possession. It takes as inputs a secret key k_1, a challenge, chal, and a proof of possession V. It returns 1 (success) if V is the correct proof of possession for the blocks determined by chal and 0 (failure) otherwise.

An auditing system can be constructed from the above algorithms and consists of two phases:

- *Setup*: The user initializes the security parameters of the system by running KeyGen. The encoded blocks are prepared as described in Section 3.1. The user then runs TagGen to generate verification metadata for each encoded block. Afterwards, both the encoded blocks and verification metadata are uploaded to the storage node. The encoded blocks are then deleted from the user's local storage. Finally, the user sends metadata needed to perform the audit to the TPA.
- *Audit*: The TPA issues an audit message, *i.e.*, a chal, to the storage node to make sure that the node correctly stores its assigned coded blocks. The node generates a proof of possession for the blocks specified in chal by running $\mathsf{GenProof}$, and it sends the possession proof back to the TPA. Finally, the TPA runs $\mathsf{VerifyProof}$ to verify the possession proof it receives.

4.2 Basic Scheme and Key Techniques

Here we describe the most basic scheme that supports remote data checking and show that it does not provide the desired properties. This basic scheme is also described in [5]. Afterwards, we describe how we improve this basic scheme to arrive at our proposed scheme.

The Basic Scheme. During the *Setup* phase, the user precomputes a message authentication code (MAC) tag, t_i, for each coded block, $\mathbf{e}_i$, using a secret key, k_1, and a standard MAC scheme, *e.g.*, HMAC. She uploads both the tags and the coded blocks to the storage node and sends k_1 to the TPA. During the *Audit* phase, to verify that the node stores $\mathbf{e}_i$ correctly, the TPA issues a request for $\mathbf{e}_i$. The node then sends $\mathbf{e}_i$ and its tag t_i to the TPA. The TPA can use k_1 and t_i to check for the integrity of $\mathbf{e}_i$. Although providing the possession checking, this scheme suffers from many drawbacks:

- It is inefficient in both computation and communication, *i.e.*, the computation and bandwidth overhead increases linearly in the number of checked blocks.
- It does not efficiently support *repair* [1], [2]: it requires the user to download all the coded blocks to be stored at the new (recovering) node then compute the verification tag for each of the block, essentially re-setting up the storage node.
- It violates privacy as the TPA learns about the blocks. Note that the straightforward way to provide privacy is to encrypt the response block using a standard encryption scheme, *e.g.*, AES. However, in this way, the TPA will not be able to verify the integrity of the original block from the provided encrypted block.

Key Techniques. We improve the basic scheme to arrive at our proposed scheme by leveraging (i) a homomorphic MAC scheme and (ii) a customized encryption scheme that exploits properties of linear network coding.

In particular, we adopt SpaceMac, a homomorphic MAC scheme that we previously designed specifically for network coding [20], [31]. We use SpaceMac to generate verification tags. With SpaceMac, the integrity of multiple blocks can be verified with the computation and communication cost of a single block verification, thanks to the ability to combine blocks and tags. SpaceMac also facilitates repair as verification metadata at the newly constructed node can be computed efficiently from existing metadata at healthy nodes.

We custom design a novel encryption scheme, called NCrypt, to protect the privacy of the response blocks. NCrypt is constructed in a way that a response block, even when encrypted, can be used by the TPA for the integrity check. NCrypt employs the andom linear combination technique of network coding to be compatible with SpaceMac verification. NCrypt is semantically secure under a chosen plaintext attack (CPA-secure). Next, we briefly describe how we use SpaceMac and describe NCrypt in detail.

4.3 The Homomorphic MAC: SpaceMac

In prior work, we designed SpaceMac and used it to combat pollution attacks in network coding [20], [21], [30], [31]. Here, we use SpaceMac to support the aggregation of file blocks and tags. SpaceMac consists of a triplet of algorithms: Mac, Combine, and Verify. The construction of SpaceMac uses a pseudo-random function (PRF) $F_1 : \mathcal{K}_1 \times (\mathcal{I} \times [1, n+m]) \rightarrow \mathbb{F}_q$, where $\mathcal{K}_1$ is the PRF key domain and $\mathcal{I}$ is the file identifier domain.

- $\mathsf{Mac}(k, \mathsf{id}, \mathbf{e}) \rightarrow t$: The MAC tag $t \in \mathbb{F}_q$ of a source block or encoded block, denoted by $\mathbf{e} \in \mathbb{F}_q^{n+m}$, under key k, can be computed by the following steps:
  – $\mathbf{r} \leftarrow (F_1(k, \mathsf{id}, 1), \cdots, F_1(k, \mathsf{id}, n+m))$.
  – $t \leftarrow \mathbf{e} \cdot \mathbf{r} \in \mathbb{F}_q$.

- $\mathsf{Combine}((\mathbf{e}_1, t_1, \alpha_1), \cdots, (\mathbf{e}_\ell, t_\ell, \alpha_\ell)) \rightarrow t$: The tag $t \in \mathbb{F}_q$ of $\mathbf{e} \stackrel{\text{def}}{=} \sum_{i=1}^{\ell} \alpha_i \mathbf{e}_i \in \mathbb{F}_q^{n+m}$ is computed as follows:

– $t \leftarrow \sum_{i=1}^{\ell} \alpha_i \, t_i \in \mathbb{F}_q$.

- Verify$(k, \mathsf{id}, \mathbf{e}, t) \rightarrow \{0,1\}$: To verify if t is a valid tag of $\mathbf{e}$ under key k, we do the following:
  – $\mathbf{r} \leftarrow (F_1(k, \mathsf{id}, 1), \cdots, F_1(k, \mathsf{id}, n+m))$.
  – $t' \leftarrow \mathbf{e} \cdot \mathbf{r}$.
  – If $t' = t$, output 1 (accept); otherwise, output 0 (reject).

Lemma 1 (Theorem 1 in [20]). *Assume that F_1 is a secure PRF. For any fixed q, n, m,* SpaceMac *is a secure (q, n, m) homomorphic MAC scheme.*

We refer the reader to [20] for the security game and proof of SpaceMac. We provide security proof of SpaceMac when used in NC-Audit in Section 6.1. If the user computes the verification tags for the source blocks using Mac, then the storage node can compute a valid MAC tag for any encoded block using Combine. The security of SpaceMac guarantees that if a block, $\mathbf{e}'$, is not a linear combination of the source blocks, then the storage node can only forge a valid MAC tag for $\mathbf{e}'$ with probability $\frac{1}{q}$. The security when using ℓ tags is improved to $\frac{1}{q^\ell}$. For clarity, we focus on a single file $\mathcal{F}$ and thus omit the file identifier id used by the above three algorithms in our subsequent discussion.

4.4 The Random Linear Encryption: NCrypt

To protect the privacy of the response file block, we need to encrypt it. The encryption, however, needs to still allow for the verification of the block. Here, we describe NCrypt, an encryption scheme that is compatible with SpaceMac. In particular, NCrypt will protect $n-1$ elements of the response block while still allowing SpaceMac integrity checking. Only $n-1$ elements rather than n is protected is because of the technical constraint needed to preserve the security guarantee of SpaceMac (shown in the proof of the subsequent Theorem 4).

Let $\bar{\mathbf{x}}$ denote the vector formed by the first $n-1$ elements of vector $\mathbf{x}$. The construction of NCrypt uses two PRFs: $F_2 : \mathcal{K}_2 \times ([1, n-1] \times [1, n-1]) \rightarrow \mathbb{F}_q$ and $F_3 : \mathcal{K}_2 \times (\{0,1\}^\lambda \times [1, n-1]) \rightarrow \mathbb{F}_q$, where $\mathcal{K}_2$ is a PRF key domain. NCrypt consists of three probabilistic polynomial time algorithms:

- Setup$(k, \bar{\mathbf{r}}) \rightarrow (p_1, \cdots, p_{n-1})$: This algorithm is run by the user to setup the encryption scheme. It takes as input a secret key k and a vector $\bar{\mathbf{r}} \neq \mathbf{0}, \bar{\mathbf{r}} \in \mathbb{F}_q^{n-1}$. It outputs $n-1$ elements in $\mathbb{F}_q$, which are called *tagging elements* and are used by the encryption. The details are as follow:
  – $\bar{\mathbf{p}}_i \leftarrow (F_2(k, i, 1), \cdots, F_2(k, i, n-1)) \in \mathbb{F}_q^{n-1}$, for $i \in [1, n-1]$.
  – $p_i \leftarrow \bar{\mathbf{r}} \cdot \bar{\mathbf{p}}_i \in \mathbb{F}_q$, for $i \in [1, n-1]$.

- Enc$(k, \bar{\mathbf{e}}, (p_1, \cdots, p_{n-1})) \rightarrow \langle \bar{\mathbf{c}}, (r, p) \rangle$: This algorithm is run by the storage node to encrypt the $n-1$ first elements of the aggregated response block. It takes as input a secret key, k, vector formed by the first $n-1$ elements of the response block, $\bar{\mathbf{e}}$, and the tagging elements, $p_1, \cdots, p_{n-1}$. It computes the encryption, $\langle \bar{\mathbf{c}}, (r, p) \rangle$, of $\bar{\mathbf{e}}$ as follows:
  – Compute $\bar{\mathbf{p}}_i, i \in [1, n-1]$, using key k as in Setup.
  – Choose r uniformly at random: $r \xleftarrow{R} \{0,1\}^\lambda$.
  – Compute the *masking coefficients*:

$$\beta_i \leftarrow F_3(k, r, i) \in \mathbb{F}_q, \text{for } i \in [1, n-1] \, .$$

  – Compute the *masking vector*:

$$\bar{\mathbf{m}} \leftarrow \sum_{i=1}^{n-1} \beta_i \, \bar{\mathbf{p}}_i \in \mathbb{F}_q^{n-1} \, .$$

  – Compute $\bar{\mathbf{c}} \leftarrow \bar{\mathbf{e}} + \bar{\mathbf{m}} \in \mathbb{F}_q^{n-1}$.
  – Compute $p \leftarrow \sum_{i=1}^{n-1} \beta_i \, p_i \in \mathbb{F}_q$.

  In essence, the data is masked with a randomly chosen vector $\bar{\mathbf{m}} \in \mathsf{span}(\bar{\mathbf{p}}_1, \cdots, \bar{\mathbf{p}}_{n-1})$.

- Dec$(k, \langle \bar{\mathbf{c}}, (r, p) \rangle) \rightarrow \bar{\mathbf{e}}$: This algorithm takes as input a secret key, k, and the cipher text, $\langle \bar{\mathbf{c}}, (r, p) \rangle$. The decryption is done as follows:
  – Compute $\bar{\mathbf{p}}_i, i \in [1, n-1]$, using key k as in Setup.
  – Compute $\beta_i \leftarrow F_3(k, r, i) \in \mathbb{F}_q$, for $i \in [1, n-1]$.
  – Compute $\bar{\mathbf{m}} \leftarrow \sum_{i=1}^{d} \beta_i \, \bar{\mathbf{p}}_i \in \mathbb{F}_q^{n-1}$.
  – Compute $\bar{\mathbf{e}} \leftarrow \bar{\mathbf{c}} - \bar{\mathbf{m}} \in \mathbb{F}_q^{n-1}$.

Theorem 2. *Assume that F_2 and F_3 are secure PRFs, then* NCrypt *is a fixed-length private-key encryption scheme for messages of length $(n-1) \times \log_2 q$ that has indistinguishable encryptions under a chosen-plaintext attack.*

Proof: Intuitively, the security of NCrypt holds because $\bar{\mathbf{m}}$ looks completely random to an adversary who observes a ciphertext $\langle \bar{\mathbf{c}}, (r, p) \rangle$ since it is computationally difficult for the adversary to compute $\bar{\mathbf{p}}_i$'s and β_i's without knowing the secret key k. The proof follows the technique used to prove the security of Construction 3.24 in [37].

We follow the notation in [37]. Denote the CPA security experiment of an encryption scheme $\Pi = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec})$ and an adversary $\mathcal{A}$ by $\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\Pi}$. The game is as follows:

- A key k is chosen uniformly at random from $\{0,1\}^\lambda$.
- The adversary $\mathcal{A}$ is given $\bar{\mathbf{r}}, p_1, \cdots, p_{n-1}$, and oracle access to Enc_k. It outputs a pair of messages $\bar{\mathbf{m}}_0$ and $\bar{\mathbf{m}}_1$, both are in $\mathbb{F}_q^{n-1}$.
- A random bit $b \leftarrow \{0,1\}$ is chosen, and then a ciphertext $c \leftarrow \mathsf{Enc}(k, \bar{\mathbf{m}}_b, (p_1, \cdots, p_{n-1}))$ is computed and given to $\mathcal{A}$. We call c the challenge ciphertext.
- The adversary $\mathcal{A}$ continues to have oracle access to Enc_k, and outputs a bit b'.
- The output of the experiment is defined to be 1 if $b' = b$, and 0 otherwise. In case $\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\Pi} = 1$, we say that $\mathcal{A}$ succeeded.

Let Π_1 be an encryption scheme that is exactly the same as Π except that a truly random function f_2 is used in place of F_2. Let $\mathsf{Adv}[\mathcal{B}, F_2]$ be the probability of an

adversary $\mathcal{B}$ with similar runtime to $\mathcal{A}$ winning the PRF security game. We have

$$|\Pr[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\Pi} = 1] - \Pr[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\Pi_1} = 1]| = \mathrm{Adv}[\mathcal{B}, F_2] \quad (1)$$

Similarly, let Π_2 be an encryption scheme that is exactly the same as Π_1 except that a truly random function f_3 is used in place of F_3. Let $\mathrm{Adv}[\mathcal{C}, F_3]$ be the probability of an adversary $\mathcal{C}$ with similar runtime to $\mathcal{A}$ winning the PRF security game:

$$|\Pr[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\Pi_1} = 1] - \Pr[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\Pi_2} = 1]| = \mathrm{Adv}[\mathcal{C}, F_3] \quad (2)$$

We claim that for every adversary $\mathcal{A}$ that makes at most $g(\lambda)$ queries to its encryption oracle (g is a polynomial function), we have

$$\Pr\left[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\Pi_2} = 1\right] \leq \frac{1}{2} + \frac{g(\lambda)}{2^{\lambda}}. \quad (3)$$

Let r_c denote the random string used when generating the challenge ciphertext. There are two cases:

(a) r_c is never used by the oracle to answer any of $\mathcal{A}$'s queries: Parse $\bar{\mathbf{e}}$ as $(e^{(1)}, \cdots, e^{(n-1)})$, $\bar{\mathbf{m}}$ as $(m^{(1)}, \cdots, m^{(n-1)})$, and $\bar{\mathbf{p}}_i$ as $(p_i^{(1)}, \cdots, p_i^{(n-1)})$. From a ciphertext returned from an oracle query, the adversary can construct the following system of equations by subtracting the query plaintext from the ciphertext:

$$\beta_1\, p_1^{(1)} + \cdots + \beta_{n-1}\, p_{n-1}^{(1)} = m^{(1)}$$
$$\cdots$$
$$\beta_1\, p_1^{(n-1)} + \cdots + \beta_{n-1}\, p_{n-1}^{(n-1)} = m^{(n-1)}$$

Note that $p_i^{(j)}$ are not all zeros w.h.p. Let β_i be unknowns, $i \in [1, n-1]$. The above system of linear equations is consistent regardless of the values of $m^{(j)}$'s since the rank of the coefficient matrix is at most $n-1$, which is the number of unknowns. Also note that the knowledge of $\bar{\mathbf{r}}$ and $p_1, \cdots, p_{n-1}$ does not contribute any additional equations w.r.t β_i to the above system. Let s be the rank of the coefficient matrix.

Now for any $w \in [1, n-1]$, assume that all $m^{(j \neq w)}, j \in [1, n-1]$, are fixed. Then $m^{(w)}$ still can take any value in $\mathbb{F}_q$ equally likely because (i) for any value of $m^{(w)}$, there is the same number of solutions, which is q^{n-1-s}, and (ii) β_j are chosen uniformly at random from $\mathbb{F}_q$. Thus, each element of the plaintext, $e^{(w)}$, is masked with a uniformly random value, $m^{(w)}$, independent of other masking elements $m^{j \neq w}, j \in [1, n-1]$. Therefore, the probability that $\mathcal{A}$ outputs $b' = b$ is exactly $1/2$, as in the case of the one-time pad.

(b) r_c is used by the oracle to answer at least one of $\mathcal{A}$'s queries: In this case, $\mathcal{A}$ may easily determine which of its messages was encrypted. This is because whenever the oracle returns a ciphertext, $\langle \hat{\mathbf{c}}, (r, p) \rangle$, it learns the masking vector $\hat{\mathbf{m}}$ associated with r since $\hat{\mathbf{m}} = \hat{\mathbf{c}} - \hat{\mathbf{e}}$. Since $\mathcal{A}$ makes at most $g(\lambda)$ queries, and r is chosen uniformly at random, the probability of this event is at most $g(\lambda)/2^{\lambda}$.

Equation (3) follows from (a) and (b). Equations (1), (2), and (3) prove the theorem. □

4.5 The Privacy-Preserving Auditing Scheme: NC-Audit

Now we are ready to describe our symmetric-key based auditing protocol, denoted by NC-Audit. In particular, NC-Audit is built from SpaceMac and NCrypt as follows:

Setup phase:

- The user divides the file into m blocks of size $n-1$ instead of n and pads to each block a random element in $\mathbb{F}_q$. This is necessary as NCrypt encrypts only the first $n-1$ elements. We still denote each padded block with its coding coefficients by $\mathbf{b}_i, i \in [1, m]$.

- The user runs KeyGen to generate MAC key, k_1, and encryption key, k_2:
  – $\mathsf{KeyGen}(1^{\lambda}) \rightarrow (k_1, k_2)$: $k_1, k_2 \xleftarrow{R} \{0,1\}^{\lambda}$.

- The user then setups the encryption scheme by computing the tagging elements, $p_1, \cdots, p_{n-1}$:
  – $\bar{\mathbf{r}} \leftarrow (F_1(k_1, 1), \cdots, F_1(k_1, n-1))$.
  – $(p_1, \cdots, p_{n-1}) \leftarrow \mathsf{Setup}(k_2, \bar{\mathbf{r}})$.

- Afterward, the user computes a tag for each source block $\mathbf{b}_i$ using Mac algorithm of SpaceMac:
  – $t_{\mathbf{b}_i} = \mathsf{Mac}(k_1, \mathbf{b}_i)$.

- The user computes MAC tags of encoded blocks using the Combine algorithm of SpaceMac. Assume $\mathbf{e} = \sum_{i=1}^{m} \alpha_i\, \mathbf{b}_i$, then its tag is computed as follows:
  – $\mathsf{TagGen}(\mathbf{e}, k_1) \rightarrow t_{\mathbf{e}} = \sum_{i=1}^{m} \alpha_i\, t_{\mathbf{b}_i}$.

- Finally, the user sends the encoded blocks, $\mathbf{e}_1, \cdots, \mathbf{e}_M$, their tags, $t_{\mathbf{e}_1}, \cdots, t_{\mathbf{e}_M}$, the tagging elements, $p_1, \cdots, p_{n-1}$, and the encryption key, k_2, to the storage node. The user also sends the coding coefficients, $\mathsf{aug}(\mathbf{e_1}), \cdots, \mathsf{aug}(\mathbf{e_M})$, and the MAC key, k_1, to the TPA. We assume that the user uses private and authentic channels to send k_1 and k_2 while using an authentic channel for sending the other data. The user then keeps the coding coefficients *for repair* and the keys but *delete* all other data.

Audit phase:

- The TPA chooses a set of indexes of blocks to be audited, $\mathcal{I} \subseteq [1, M]$, and chooses the coefficients for these blocks uniformly at random: $\alpha_i \xleftarrow{R} \mathbb{F}_q, i \in \mathcal{I}$. The challenge includes the indexes of the blocks and their corresponding coefficients:
  – Prepare $\mathsf{chal} = \{(i, \alpha_i) \,|\, i \in \mathcal{I}\}$.

- GenProof run by the storage node to generate the proof of storage, V, is implemented as follows:
  – Compute the aggregated block:
  $\hat{\mathbf{e}} = \sum_{i \in \mathcal{I}} \alpha_i\, \hat{\mathbf{e}}_i$. Parse $\hat{\mathbf{e}}$ as $(\bar{\mathbf{e}}, e^{(n)})$.
  – Compute the aggregated tag:
  $t = \sum_{i=\in \mathcal{I}} \alpha_i\, t_{\mathbf{e}_i}$.
  – Encrypt the response block:

$\langle \bar{\mathbf{c}}, (r,p) \rangle \leftarrow \mathsf{Enc}(k_2, \bar{\mathbf{e}}, (p_1, \cdots, p_{n-1}))$.
The node then sends $V = (\langle \bar{\mathbf{c}}, (r,p) \rangle, e^{(n)}, t)$ back to the TPA.

- VerifyProof run by the TPA to verify the proof V is implemented as follows:
  – Compute coefficients of $\hat{\mathbf{e}}$:
  $\mathsf{aug}(\mathbf{e}) = \sum_{i \in \mathcal{I}} \alpha_i \, \mathsf{aug}(\mathbf{e_i})$.
  – Let $\mathbf{c} = (\bar{\mathbf{c}} \,|\, e^{(n)} \,|\, \mathsf{aug}(\mathbf{e}))$, where "|" denotes augmentation. Return result of $\mathsf{Verify}(k_1, \mathbf{c}, t + p)$.

Correctness. The correctness of NC-Audit is guaranteed by the following theorem. Its security is proved in Section 6.

Theorem 3. *If the storage node follows* NC-Audit *and computes the aggregated response block using uncorrupted blocks, then the TPA will accept the proof.*

Proof: Let $\mathbf{r} = (F_1(k, \mathsf{id}, 1), \cdots, F_1(k, \mathsf{id}, n+m))$. Note that

$$\begin{aligned} \mathbf{c} &= (\bar{\mathbf{c}} \,|\, e^{(n)} \,|\, \mathsf{aug}(\mathbf{e})) = ((\bar{\mathbf{e}} + \bar{\mathbf{m}}) \,|\, e^{(n)} \,|\, \mathsf{aug}(\mathbf{e})) \\ &= \mathbf{e} + (\bar{\mathbf{m}} \,|\, 0, \cdots, 0)\,. \end{aligned}$$

Thus, in the Verify,

$$\begin{aligned} t' &= \mathbf{c} \cdot \mathbf{r} = \mathbf{e} \cdot \mathbf{r} + \bar{\mathbf{m}} \cdot \bar{\mathbf{r}} \\ &= t + \sum_{i=1}^{n-1} \beta_i \, \bar{\mathbf{p}}_i \cdot \bar{\mathbf{r}} = t + \sum_{i=1}^{n-1} \beta_i \, p_i = t + p\,. \end{aligned}$$

Therefore, Verify returns 1. Hence, the TPA accepts the proof. □

5 Repair and Data Dynamics

Here, we discuss how NC-Audit efficiently supports the repair of a failed node as well as changes to the data made by the user.

5.1 Support for Node Repair

When there is a node failure, the user creates a new node to replace this node. Based on the coding coefficients of the coded blocks at the remaining healthy nodes, the user instructs the healthy nodes to send appropriate coded blocks to the new node. The new node then linearly combines them, according to the user instruction, to construct its own coded blocks. This new node may construct the same coded blocks that the failed node had (*exact repair*), or completely different coded blocks (*functional repair*) [1].

Using NC-Audit, the verification tags of the newly constructed blocks at the new node do not need to be computed by the user. In particular, the healthy nodes can send along the verification tags of the coded blocks that they send to the new node. The new node can use Combine to generate tags corresponding to the coded blocks that it needs to construct. Finally, the user sends the coding coefficients of the coded blocks at the newly constructed node to the TPA so that it can audit this new node. As a result, with NC-Audit, there is negligible cost, in term of both bandwidth and computation of verification metadata, to the user when repairing a failed node. This stands in stark contrast with the prior integrity checking scheme for NC-based storage [18], which requires the user to download many data blocks (equal to the repair bandwidth) and compute security metadata for the newly coded blocks herself.

Last but not least, since the TPA audits the new node based on the new set of coefficients, a malicious node cannot carry out a *replay attack* [18] (discussed in Section 2.2); otherwise, it will not pass the audit. Furthermore, we assume that the healthy remaining nodes send valid data and tags to the new node. If there is a malicious node that sends corrupted data or tags, the storage systems is considered polluted. Dealing with pollution attacks is out of the scope of this paper; we refer the reader to previous work which explicitly combat pollution attacks [21], [25]–[28], [30], [31], [35], [38].

5.2 Support for Data Dynamics

Next, we discuss how NC-Audit supports changes that the user may want to make to their outsourced data, including block update, append, insert, and delete – with the first two operations generally considered the most important operations for NC-based storage. We stress that how each change is carried out by the storage cloud is dependent on the coding scheme used by the cloud as well as how the cloud is designed. In other words, the changes of data itself are orthogonal to and out of the scope of this work. Similar to [12], we focus on how the security metadata can be maintained correctly and efficiently when using our scheme, regardless of how the data is changed.

Block Update. Assume the user wants to update the source block, $\mathbf{b}_j$, for some $j \in [1, m]$. Denote the new block after the update $\mathbf{b}'_j$. It first needs to learns the tag of $\mathbf{b}_j$, which can be done as follows: Assume $\mathbf{b}_j = \sum_{i=1}^{m} \alpha_i \, \mathbf{e}_i$, then $t_{\mathbf{b}_j} = \sum_{i=1}^{m} \alpha_i \, t_{\mathbf{e}_i}$. For $i \neq 0$, the user can download $t_{\mathbf{e}_i}$ from the appropriate storage nodes to compute $t_{\mathbf{b}_j}$.

The user then computes the tag $t_{\mathbf{b}'_j}$ of $\mathbf{b}'_j$ under key k_1 using Mac. Finally, it sends $t_{\mathbf{b}'_j}$ and $t_{\mathbf{b}_j}$ to the TPA using an authentic and secure channel. Subsequently, whenever challenging a storage node and obtaining a response block which involves $\alpha_j \, \mathbf{b}_j$, the TPA runs VerifyProof with the tag $t + \alpha_j (t_{\mathbf{b}'_j} - t_{\mathbf{b}_j})$ instead of t. To see why this is the case, let $\hat{\mathbf{e}} = \alpha_j \, \hat{\mathbf{b}}_j + \sum_{i=1,\cdots,M; i \neq j} \alpha_i \, \hat{\mathbf{b}}_i$ be the aggregated response block (before encryption). Its corresponding tag that is sent back with the proof of possession is $t = \alpha_j \, t_{\mathbf{b}_j} + \sum_{i=1,\cdots,M; i \neq j} \alpha_i \, t_{\mathbf{b}_i}$. But since $\mathbf{b}_j$ is now updated, the correct tag must be $t' = \alpha_j \, t_{\mathbf{b}'_j} + \sum_{i=1,\cdots,M; i \neq j} \alpha_i \, t_{\mathbf{b}_i}$. Note that if $\hat{\mathbf{e}}$ is not updated correctly then by the security guarantee of SpaceMac, w.h.p. t' is not a valid tag for $\mathbf{e}$. Subsequent updates to this j-th block can be carried out similarly but without recomputing $t_{\mathbf{b}_j}$.

This approach requires the TPA to store a verification tag for every updated source block, which is $O(m)$. This overhead is negligible compared to the outsourced data $O((n+m)MN))$, where $n \gg m$. Finally, we assume that the storage nodes send back correct tags. If one wants to consider a stronger threat model where the storage nodes may send back bogus tags, then there are two possible solutions: (i) modifying the auditing scheme to require the user to store the source tags, $t_{\mathbf{b}_j}$; in this case, the additional client storage overhead is $O(m)$; or (ii) a traditional MAC scheme computed on the coding coefficient, $\mathsf{aug}(\mathbf{e_i})$, and verification tag, $t_{\mathbf{e}_i}$, can be used to protect the integrity of the tag.

Block Append. Assume that the user wants to append a source block, $\hat{\mathbf{b}}_*$, to the system. The encoded $\mathbf{b}_*$ has the following form:

$$\mathbf{b}_* = (\overbrace{—\hat{\mathbf{b}}_*—}^{n}, \overbrace{0, \cdots, 0}^{m}, 1) \in \mathbb{F}_q^{n+m+1} .$$

The user first computes the tag $t_{\mathbf{b}_*}$ of $\mathbf{b}_*$ under k_1 using Mac (now for vectors with size $n+m+1$) as follows:

– $\mathbf{r} \leftarrow (F_1(k,1), \cdots, F_1(k, n+m+1))$.

– $t_{\mathbf{b}_*} \leftarrow \mathbf{b}_* \cdot \mathbf{r} \in \mathbb{F}_q$.

It then sends $t_{\mathbf{b}_*}$ to all storage nodes that have coded packets that involve $\mathbf{b}_*$.

Note that when an append happens, the vector representation of a previous source block, $\mathbf{b}_i, i \in [1,m]$, is appended with a zero. However, its verification tag, computed using Mac, remains the same since $0 \times F_1(k, n+m+1) = 0$. Consequently, for coded packets that do not involve $\mathbf{b}_*$, their tags remain the same: if $\mathbf{e} = \sum_{i=1}^{m} \alpha_i \mathbf{b}_i$, then its new tag equals is old tag: $t'_{\mathbf{e}} = t_{\mathbf{e}} = \sum_{i=1}^{m} \alpha_i t_{\mathbf{b}_i}$. For coded packets that involve $\mathbf{b}_*$, the storage node can compute their new tags using $t_{\mathbf{b}_*}$: assume α_* of $\mathbf{b}_*$ is added to $\mathbf{e}$, then $t_{\mathbf{e}'} = t_{\mathbf{e}} + \alpha_* t_{\mathbf{b}_*}$.

Afterwards, the user must send the new coding coefficients, $\mathsf{aug}(\mathbf{e_1}), \cdots, \mathsf{aug}(\mathbf{e_M})$, to the TPA. Note that how the system updates its set of coding coefficients depends on how the underlining coding scheme handles block append. Finally, since the TPA carries out audits using this new set of coefficients, if the storage node does not update its data and tag correctly, it will not pass the subsequent audits. In particular, since the TPA computes $\mathsf{aug}(\mathbf{e})$ in VerifyProof locally, if the response block $\hat{\mathbf{e}}$ (before encryption) is not updated correctly, in the proof of Theorem 3, $\mathbf{c} \neq \mathbf{e} + (\bar{\mathbf{m}} \,|\, 0, \cdots, 0)$. Thus, by the security guarantee of SpaceMac, VerifyProof will fail w.h.p.

Block Insert. Assume that the user wants to insert a new source block, $\mathbf{b}_*$, before a source bock $\mathbf{b}_j, j \in [1,m]$. After an insertion, a previous source block, $\mathbf{b}_i, i \geq j$, will have the following form:

$$\mathbf{b}_i = (\overbrace{—\hat{\mathbf{b}}_i—}^{n}, \underbrace{0, \cdots, 0, 1, 0, \cdots, 0}_{i+1}^{m+1}) \in \mathbb{F}_q^{n+m+1} .$$

Since $\mathbf{b}_i$'s coefficient 1 is shifted to the right by 1, its new tag computed by Mac no longer equals its old tag:

$$\hat{\mathbf{r}} \cdot \hat{\mathbf{b}}_i + F_1(k, i+1) \neq \hat{\mathbf{r}} \cdot \hat{\mathbf{b}}_i + F_1(k,i) .$$

As a result, a straightforward insertion does not work.

To this end, we take an approach similar to [12], where a block insert is implemented with a block append and a mapping. In particular, the block is first appended to the system using *Block Append* above; then the user needs to keep a mapping of the index of the appended block to its appropriate position. This requires user storage which is linear in the number of blocks inserted.

Block Delete. We assume that the number of blocks to be deleted is small relatively to the file size. If a large portion of the file is to be deleted then it is best to rerun the *Setup* phase of NC-Audit. Similar to [12], we consider deletion of a block as changing it to a special block. Thus, updating the metadata to reflect the deletion can be done as in the *Block Update* case.

In summary, when using NC-Audit, the user can update integrity metadata very efficiently to support data dynamics, *i.e.*, without downloading data blocks.

6 Security Analysis

6.1 Data Possession Guarantee

When using SpaceMac in NC-Audit, some information about $\mathbf{r}$ in the SpaceMac construction are available to the adversary. In particular, the storage node knows the following $n-1$ equations: $\bar{\mathbf{p}}_i \cdot \bar{\mathbf{r}} = p_i, i \in [1, n-1]$. The following theorem states that even when these $n-1$ equations are exposed, SpaceMac is still a secure homomorphic MAC.

Theorem 4. *Assume that F_1 is a secure PRF. For any fixed q, n, m, assume that a probabilistic polynomial time adversary $\mathcal{A}$ knows any $n-1$ linearly independent vectors, $\bar{\mathbf{p}}_1, \cdots, \bar{\mathbf{p}}_{n-1}$, and any $n-1$ constants, $p_1, \cdots, p_d$, such that $\bar{\mathbf{p}}_i \cdot \bar{\mathbf{r}} = p_i$, where $\mathbf{r}$ is used in the construction of* SpaceMac. *The probability that $\mathcal{A}$ wins the* SpaceMac *security game, denoted by* $\mathrm{Adv}[\mathcal{A}, \mathsf{SpaceMac}]$, *is at most*

$$\text{PRF-Adv}[\mathcal{B}, F_1] + \frac{1}{q} ,$$

where $\text{PRF-Adv}[\mathcal{B}, F_1]$ *is the probability of an adversary $\mathcal{B}$ with similar runtime to $\mathcal{A}$ winning the PRF security game.*

Proof: The security game, called the Attack Game 1, of SpaceMac involves a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$, and is as follows:

- *Setup.* $\mathcal{C}$ generates a random key $k \xleftarrow{R} \mathcal{K}$
- *Queries.* $\mathcal{A}$ adaptively queries $\mathcal{C}$, where each query is of the form $(\mathrm{id}, \mathbf{y})$. For each query, $\mathcal{C}$ replies to $\mathcal{A}$ with the corresponding tag $t \leftarrow \mathsf{Mac}(k, \mathrm{id}, \mathbf{y})$.
- *Output.* $\mathcal{A}$ eventually outputs a tuple $(\mathrm{id}^*, \mathbf{y}^*, t^*)$.

Up to the time $\mathcal{A}$ outputs, it has queried $\mathcal{C}$ multiple times. Let l denote the number of times $\mathcal{A}$ queried $\mathcal{C}$ using id^* and get tags for l vectors, $\mathbf{y}_1^*, \cdots, \mathbf{y}_l^*$, of these queries.

We consider that the adversary wins the security game if and only if

- $(y_*^{(n+1)}, \cdots, y_*^{(n+m)}) \neq \mathbf{0}$ (trivial forge otherwise),
- $\mathsf{Verify}(k, \mathrm{id}^*, \mathbf{y}^*, t^*) = 1$, and
- $\mathbf{y}^* \notin \mathsf{span}(\mathbf{y}_1^*, \cdots, \mathbf{y}_l^*)$.

Here, we prove Theorem 4 with respect to a slightly different security game, called Attack Game 2. This Attack Game 2 is similar to Attack Game 1, except that in the *Queries* phase, for each distinct id, the space spanned by the vectors used in the queries has dimension at most m. This Attack Game 2 is stricter but better fits the reality: since the dimension of the source space Π is only m, the adversary must only learn tags of vectors in spaces having dimensions at most m.

The proof is done by using a sequence of games denoted Game 0 and Game 1. Let W_0 and W_1 denote the events that $\mathcal{A}$ wins the homomorphic MAC security in Game 0 and Game 1, respectively. Game 0 is identical to Attack Game 2 applied to the scheme SpaceMac. Hence,

$$\Pr[W_0] = \mathrm{Adv}[\mathcal{A}, \mathsf{SpaceMac}] \tag{4}$$

Game 1 is identical to Game 0 except that the challenger $\mathcal{C}$ computes $\mathbf{r} \leftarrow (r_1, \cdots, r_{n+m})$, where r_i is chosen uniformly at random from $\mathbb{F}_q$: $r_i \xleftarrow{R} \mathbb{F}_q$ instead of $r_i \leftarrow F(k, \mathrm{id}, i)$, and everything else remains the same. Then, there exists a PRF adversary $\mathcal{B}$ such that

$$|\Pr[W_0] - \Pr[W_1]| = \text{PRF-Adv}[\mathcal{B}, F] \tag{5}$$

The complete challenger in Game 1 works as follows:

Queries. $\mathcal{A}$ adaptively queries $\mathcal{C}$, where each query is of the form $(\mathrm{id}, \mathbf{y})$. If id is already used in m previous query, $\mathcal{C}$ discards the query. Otherwise, $\mathcal{C}$ replies to query i of $\mathcal{A}$ as follows:

if id is never used in any of the previous queries:
  $\mathbf{r}_i := (r_1^i, \cdots, r_{n+m}^i)$, where $r_j^i \xleftarrow{R} \mathbb{F}_q, j \in [n+m]$
else:
  $\mathbf{r}_i :=$ the one used in the previous response
send $t := \mathbf{y}_i \cdot \mathbf{r}_i$ to $\mathcal{A}$

Output. $\mathcal{A}$ eventually outputs a tuple $(\mathrm{id}^*, \mathbf{y}^*, t^*)$. When $\mathbf{y}^*$ does not equal $\mathbf{0}$, to determine if $\mathcal{A}$ wins the game, we compute

if $\mathrm{id}^* = \mathrm{id}_i$ (for some i) then // case (i)
  set $\mathbf{r}^* := \mathbf{r}_i$
else // case (ii)
  set $\mathbf{r}^* := (r_1^*, \cdots, r_{n+m}^*)$, where $r_i^* \xleftarrow{R} \mathbb{F}_q, i \in [n+m]$

Let l denote the number of times $\mathcal{A}$ queried $\mathcal{C}$ using id^* and get tags for l vectors, $\mathbf{y}_1^*, \cdots, \mathbf{y}_l^*$, of these queries. The adversary wins the game, *i.e.*, event W_1 happens, if and only if

$$t^* = \mathbf{y}^* \cdot \mathbf{r}^*, \text{ and} \tag{6}$$
$$\mathbf{y}^* \notin \mathsf{span}(\mathbf{y}_1^*, \cdots, \mathbf{y}_l^*). \tag{7}$$

We will show that $\Pr[W_1] = \frac{1}{q}$. Let T be the event that $\mathcal{A}$ outputs a tuple with a completely new id^*, *i.e.*, $\mathcal{A}$ never made queries using id^* before.

- When T happens, *i.e.*, in case (ii), since r_i^*'s are indistinguishable from random values and $(y_*^{(n+1)}, \cdots, y_*^{(n+m)}) \neq \mathbf{0}$, the right hand side of equation (6) is a completely random value in $\mathbb{F}_q$. Thus,

$$\Pr[W_1 \wedge T] = \frac{1}{q} \Pr[T]. \tag{8}$$

- When T does not happen, *i.e.*, in case (i): $\mathbf{r}^*$ of equation (6) equals $\mathbf{r}_i$ for some i, and $\mathbf{r}^*$ has been used to generate tags for vectors $\mathbf{y}_1^*, \cdots, \mathbf{y}_l^*$. In this case, we proceed by showing that for a fixed $\mathbf{y}^*$, t^* looks indistinguishable from a random value in $\mathbb{F}_q$. Let $\Pi^* = \mathsf{span}(\mathbf{y}_1^*, \cdots, \mathbf{y}_l^*)$, which has dimension $\delta \leq m$. Let $\{\mathbf{b}_1, \cdots, \mathbf{b}_\delta\}$ be a basis of Π^*. Let $r_1^*, \cdots, r_{n+m}^*$ be the unknowns. The given prior knowledge, the queries, and the output form the following system of linear equations: $\bar{\mathbf{p}}_1 \cdot \bar{\mathbf{r}}^* = p_1; \cdots; \bar{\mathbf{p}}_d \cdot \bar{\mathbf{r}}^* = p_{n-1}; \mathbf{y}_1^* \cdot \mathbf{r}^* = t_{\mathbf{y}_1^*}; \cdots; \mathbf{y}_l^* \cdot \mathbf{r}^* = t_{\mathbf{y}_l^*}; \mathbf{y}^* \cdot \mathbf{r}^* = t^*$.

This system is equivalent to the following system: $\bar{\mathbf{p}}_1 \cdot \bar{\mathbf{r}}^* = p_1; \cdots; \bar{\mathbf{p}}_d \cdot \bar{\mathbf{r}}^* = p_{n-1}; \mathbf{b}_1 \cdot \mathbf{r}^* = t_{\mathbf{b}_1}; \cdots; \mathbf{b}_\delta \cdot \mathbf{r}^* = t_{\mathbf{b}_\delta}; \mathbf{y}^* \cdot \mathbf{r}^* = t^*$. Note that $t_{\mathbf{b}_i}$ is a linear combination of some $t_{\mathbf{y}_i^*}$. Without loss of generality, assume $\bar{\mathbf{p}}_i$ and $\mathbf{b}_i$ are linearly independent (otherwise, the number of equations the adversary learns would be less). Since the coefficients of $\mathbf{y}^*$ are not all zeros and $\mathbf{y}^* \notin \Pi^*$, $\mathbf{y}^*$ is linearly independent of $\bar{\mathbf{p}}_i$ and $\mathbf{b}_i$. Thus, the above system of $n+m$ unknowns is consistent regardless of the value of t^* because the coefficient matrix has rank $n+\delta$, which equals the number of equations. Furthermore, for any value t^*, the solution space always has the same size $q^{m-\delta}$. Thus, for a fixed $\mathbf{y}^*$, its valid tag t^* could be any value in $\mathbb{F}_q$ equally likely, given that r_i^*'s are chosen uniformly at random from $\mathbb{F}_q$. As a result, the probability that the adversary chooses a correct t^* is $1/q$. Thus,

$$\Pr[W_1 \wedge \neg T] = \frac{1}{q} \Pr[\neg T]. \tag{9}$$

- From equations (8) and (9), we have

$$\Pr[W_1] = \Pr[W_1 \wedge T] + \Pr[W_1 \wedge \neg T] = \frac{1}{q}. \tag{10}$$

Equations (4), (5), and (10) together prove the theorem. □

Now, we are ready to prove the data possession guarantee of NC-Audit.

Theorem 5. *With probability at least $1 - \frac{2}{q}$, the storage node can pass a check if and only if it possesses the blocks specified in the challenge of the check.*

Proof: Theorem 3 shows that if the storage node possesses the data then it can pass the check. It remains to show that if the node passes the check then it possesses the corresponding blocks w.h.p. Let us prove the converse, *i.e.*, if there are corrupted or missing blocks, the node will fail the check w.h.p.

For simplicity, we assume that when responding to a challenge involving a block that no longer exists in the storage, the node replaces it with a block chosen

uniformly at random in $\mathbb{F}_q^{n+m}$. Denote the correct, unencrypted aggregated block by $\mathbf{e}$, *i.e.*, $\mathbf{e} = \sum_{i\in\mathcal{I}} \alpha_i \mathbf{e}_i$. Denote the data of the response block actually computed by the storage node by $\hat{\mathbf{a}}$ and denote $(\hat{\mathbf{a}} \,|\, \mathsf{aug}(\mathbf{e}))$ by $\mathbf{a}$.

If there is at least one error in the data of one of the block or there is at least one missing block, then $\mathsf{Prob}[\hat{\mathbf{a}} = \hat{\mathbf{e}}] \leq \frac{1}{q}$ because α's are chosen uniformly at random from $\mathbb{F}_q$. Note that $\mathbf{e}$ is in the source space: $\mathbf{e} \in \Pi$, thus if $\hat{\mathbf{a}} \neq \hat{\mathbf{e}}$ then $\mathbf{a} \notin \Pi$. Therefore, $\mathsf{Prob}[\mathbf{a} \in \mathbf{\Pi}] = \mathsf{Prob}[\mathbf{a} = \mathbf{e}] \leq \frac{1}{q}$ (a).

Furthermore, the security of SpaceMac from Theorem 4 guarantees that the node can provide a valid tag of $\mathbf{a} \notin \Pi$ with probability at most $\frac{1}{q}$. Finally, without loss of generality, we can ignore the encryption because if the node already knows a valid tag of $\mathbf{a}$, it can provide the correct encryption to pass the check. Meanwhile, if the node does not know a valid tag of $\mathbf{a}$, its chance of forging a valid tag for the cipher text $\mathbf{c}$ is still bounded by the security guarantee of SpaceMac, which is at most $\frac{1}{q}$ (b).

As a result, from (a) and (b), the probability of passing the check when there is error or missing block is at most $\frac{2}{q}$. □

NC-Audit actually provides a stronger data possession guarantee. It ensures that the user can extract the data stored on the storage node just by collecting response of the node from the checking protocol. We provide proof of retrievability based on the theoretical framework of [13], which is derived from [10] and [9].

Theorem 6. *Assume that the storage node responses correctly to a fraction $1-\epsilon$ of challenge uniformly, where $\epsilon < \frac{1}{2}$. The user can extract $\mathbf{e}_1, \cdots, \mathbf{e}_M$ by performing γ challenge-response interactions with the storage node with high probability (depending on γ, ϵ, and q).*

Proof: Theorem 5 implies that if a node responses correctly to a fraction of challenge, then with probability at least $1 - \frac{2}{q}$, the response block is a correct linear combination of the blocks stored at the node. For a challenge coefficient vector $(\alpha_1, \cdots, \alpha_M)$, the user can challenge the node using a number of constant-multiples of the vector, *e.g.*, $(c\,\alpha_1, \cdots, c\,\alpha_M)$ for some constant c, to learn the responses (including incorrect responses), and then use majority decoding to learn the correct equation $\sum_{i=1}^{M} \alpha_i \mathbf{e}_i = \mathbf{d}$, where $\mathbf{d}$ is some constant vector. By collecting M linearly independent equations of this form, the user can solve for $\mathbf{e}_1, \cdots, \mathbf{e}_M$ using Gaussian elimination.

Note that for a fixed $\epsilon < \frac{1}{2}$, the probability of learning one correct equation depends on both q and the number of queries made using the multiples of the corresponding coefficient vector. For a fixed q, this probability can be made arbitrarily high by increasing the number of queries. □

6.2 Privacy-Preserving Guarantee

We summarize the privacy guarantee of NC-Audit in the following theorem.

Theorem 7. *From the response of the storage node, the TPA does not learn any information about the outsourced data, except for the information that could be derived from the MAC tag.*

Proof: The claim is a direct consequence of Theorem 2 and the fact that the padding element is chosen randomly. □

We stress that the information derived from the MAC tags are not sufficient to derive the outsourced data. To be concrete, each tag is a weighted sum of symbols belonging to the same block. Also, the outsourced data consists of $m \times n$ field symbols, which could be considered as unknowns of a system of linear equations, and the knowledge given by the tags and the MAC key only gives at most n linearly independent equations.

7 Performance Evaluation

7.1 Client Storage Overhead

NC-Audit requires the user and the TPA to store the coding coefficients, which is in $O(mMN)$ space. The user needs the coefficients to carry out repair and block update, while the TPA needs the coefficients to carry out audits. In any case, this overhead is orders of magnitude less than the outsourced data, which is in $O((n+m)MN)$ space, since $n \gg m$. In fact, in a practical NC storage cloud, the storage needed to store the coding coefficients could be kept less than 16 B (*i.e.*, constant storage) while being able to support arbitrary file size [24]. Table 1 compares client storage overhead of NC-Audit and other recent schemes [11], [15], [18].

7.2 Bandwidth Overhead

Integrity Checking. For each audit round, the major communication cost is the cost of sending the proof of possession from the storage node to the TPA, which is dominated by the size of the (encrypted) data bock. Thanks to homormophic property of SpaceMac, blocks in the challenge can be aggregated. We achieve similar bandwidth overhead compared to prior schemes for integrity checking of cloud data [9], [11], [15], [18], *i.e.*, the proof of possession for multiple blocks contains only a single block (of size varying from 4 KB [5] to 1.6 MB [18]).

We note that a coding scheme can be modified to support small block size by subdividing source blocks. For instance, to halve the size of a block, each source block $\hat{\mathbf{b}}_i$ can be divided into two equal blocks $\hat{\mathbf{b}}_{i,1}$ and $\hat{\mathbf{b}}_{i,2}$. The global coefficients of the blocks are then changed as follow:

$$\mathbf{b}_{i,1} = (—\hat{\mathbf{b}}_{i,1}—, \underbrace{0, \cdots, 0, 1}_{2i-1}, \overbrace{0, \cdots, 0}^{2m}) \in \mathbb{F}_q^{n+2m},$$

$$\mathbf{b}_{i,2} = (—\hat{\mathbf{b}}_{i,2}—, \underbrace{0, \cdots, 0, 1}_{2i}, \overbrace{0, \cdots, 0}^{2m}) \in \mathbb{F}_q^{n+2m}.$$

The coding scheme is kept the same: the coding operations performed on $\mathbf{b}_i$ are translated to similar coding operations done on both $\mathbf{b}_{i,1}$ and $\mathbf{b}_{i,2}$. Note however that the overhead of the coefficients is doubled in this case. In general, it increases linearly in the number of source blocks.

Repairing and Updating. As shown in Section 5, when using NC-Audit, the user does not need to download any data block to repair failed node or update the outsourced data. In contrast, in [18], the user needs to download an amount of data equal to the repair bandwidth to setup integrity metadata for the new coded blocks herself.

Encryption. The amount of additional bandwidth to support encryption is small. In particular, NCrypt requires the storage node to send with the encrypted block, $\bar{\mathbf{c}}$, the random value, r, of size λ (typically 80 bits [5]), the auxiliary tag p, and the random padding element $e^{(n)}$, which are both of size $\log_2 q$. These are negligible compared to the block size: $n \log_2 q$ (0.3% for $q = 2^8, n = 4 \times 2^{10}$).

7.3 Computational Overhead

We first analyze the cost of each operation in NC-Audit by the number of finite filed multiplications involved, which is the dominating cost factor. We then present the cost of each operation from our real implementation in Java. We omit the cost of computing PRF values that do not take as input random seeds since they can be precomputed.

Integrity Checking with Encryption:

1. Storage Node Overhead: In NC-Audit, the cost to compute the proof of possession includes the cost to compute (i) the aggregated response block, $\bar{\mathbf{e}}$, (ii) the response tag, t, (iii) the masking vector, $\bar{\mathbf{m}}$, and the auxiliary element, p. The total cost is dominated by the cost to compute $\bar{\mathbf{e}}$ and $\bar{\mathbf{m}}$. $\bar{\mathbf{m}}$ can be precomputed in advance as it is independent of the challenge. Let C be the average number of blocks specified in a challenge. The average cost to compute a response per challenge is $C \times n$ multiplications with precomputations of $\bar{\mathbf{m}}$ and $(C + n - 1) \times n$ without.

2. TPA Overhead: In NC-Audit, verifying a proof of possession can be done very efficiently. In particular, the cost to verify include the time to (i) compute the coefficients of the response block and (ii) run the Verify of SpaceMac. Let ℓ be the number of tags used. The total cost is $C \times m + \ell \times (n + m)$ multiplications.

Repairing and Updating:

As described in Section 5, repairing a failed node does not incur any computation cost at the user side. Updating a block also incurs very small amount of computational overhead by the user. In particular, the dominant cost is due to computation of the tag of the new block (either to be updated, inserted, or appended), which entails $n + m$ field multiplications.

Implementation:

We implement NC-Audit in Java to compare its performance with recent schemes [11], [15], [18]. For a fair of comparison with [11], [15], we use $q = 2^8$ and $\ell = 10$ to provide 80-bit security, and we also set block size to 4 KB ($n = 4 \times 2^{10}$), $m = 500$, and the number of blocks indicated by a challenge to $C = 300$. We stress that the choice of parameters may be different in a practical NC storage system, *e.g.*, in [24], a block size could be as big as 4 MB while the storage space taken by the coefficients could be kept below 16 B.

We implement finite field multiplications in $\mathbb{F}_{2^8}$ by table look-ups and additions using XORs. We also precomputed values that do not depend on the challenges.

Table 1 compares both the bandwidth overhead and computational overhead of different remote data integrity checking schemes. The reported numbers for [15] and [11] are taken from [11]. (The overhead of the scheme in [15] is similar to the public-key based scheme in [9].) We refer the reader to [11] for the detailed setup. We implement the checking scheme in [18] ourselves. For this scheme, we use AES with CBC mode from Java *crypto* library to decrypt coefficients. We refer the reader to Appendix A in [18] for the detailed description of this scheme. The number reported for NC-Audit and the scheme in [18] are the average of 100 runs on a computer with 2.8 Ghz CPU and 32 GB RAM.

Table 1 shows that NC-Audit manages to achieve top bandwidth efficiency, and at the same time, having very small computational overhead. The computational overhead of NC-Audit is orders of magnitude smaller than those of [15] and [11]. This is due to the fact that NC-Audit is symmetric-key based while the schemes in [15] and [11] are public-key based and make heavily use of expensive bilinear mapping operations. We also note that the scheme in [18] achieves similar storage node computational overhead as it is also symmetric-key based; however, due to the cost of executing $C \times m = 150,000$ numbers of decryption for the coefficients, the computational overhead of the TPA is much larger, in the order of seconds.

8 Conclusion

In this work, we propose NC-Audit, a remote data integrity checking scheme for NC-based storage systems. NC-Audit is built based on a homomorphic MAC scheme custom made for network coding, SpaceMac, and a novel CPA-secure encryption scheme, NCrypt. NC-Audit allows for efficient integrity checking, supports repair of failed node and data dynamics (including block update, append, insert, and delete), and prevents leakage of the outsourced data when the audit is done by a third party.

References

[1] A. G. Dimakis, K. Ramchandran, Y. Wu, and C. Suh, "A Survey on Network Codes for Distributed Storage," *Proceedings of the IEEE*, vol. 99, no. 3, pp. 476–489, Mar. 2011.

		Wang 2009 [15]	Wang 2010 [11]	Chen 2010 [18]	NC-Audit
Features		Public-Key Audit	Public-Key Audit	Private-Key Audit	Private-Key Audit
		No NC Repair	*No* NC Repair	NC Repair	*Efficient* NC Repair
		Data Dynamics	*No* Data Dynamics	*No* Data Dynamics	Data Dynamics
		No Audit Privacy	Audit Privacy	*No* Audit Privacy	Audit Privacy
Client Storage	Audit Overhead	$O(1)$	$O(1)$	$O(1)$	$O(mMN)$
	Repair Overhead	N/A	N/A	$O(mMN)$	$O(mMN)$
Bandwidth	Audit Overhead	1 block	1 block	1 block	1 block
	Repair Overhead	N/A	N/A	repair bandwidth	0*
	Updating Overhead	0*	N/A	N/A	0*
	Enc. Overhead	N/A	0*	N/A	0*
Computation	Security	80-bit			
	Parameters	300 blocks per challenge, 4 KB block size			
	Testbed Config.	1.86 Ghz CPU, 2GB RAM		2.8 Ghz CPU, 32 GB RAM	
	Server Overhead	270 ms	273 ms	3.19 ms	4.69 ms
	Auditor Overhead	491 ms	493 ms	2.76 s	0.73 ms

TABLE 1
Comparisons of different remote data integrity checking schemes. 0* indicates no data block needs to be downloaded by the user to support the feature. N/A means not applicable due to the lack of support.

[2] A. Dimakis, B. Godfrey, Y. Wu, M. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," *IEEE Transactions on Information Theory*, vol. 56, no. 9, pp. 4539–4551, Sep. 2010.

[3] A. Krioukov, L. N. Bairavasundaram, G. R. Goodson, K. Srinivasan, R. Thelen, A. C. Arpaci-dusseau, and R. H. Arpaci-dusseau, "Parity Lost and Parity Regained," in *USENIX Conference on File and Storage Technologies (FAST)*, San Jose, CA, Feb. 2008, pp. 127–141.

[4] B. Schroeder, S. Damouras, and P. Gill, "Understanding latent sector errors and how to protect against them," in *USENIX Conference on File and Storage Technologies (FAST)*, San Jose, CA, Sep. 2010, pp. 1–23.

[5] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable data possession at untrusted stores," in *ACM Conference on Computer and Communication Security (CCS)*, Alexandria, VA, Oct. 2007, pp. 598–609.

[6] M. A. Shah, R. Swaminathan, and M. Baker, "Privacy-Preserving Audit and Extraction of Digital Contents," in *Cryptology ePrint Archive, Report 2008/186*, 2008. [Online]. Available: http://eprint.iacr.org/2008/186.pdf

[7] C. Wang, Q. Wang, K. Ren, and W. Lou, "Ensuring Data Storage Security in Cloud Computing," in *International Workshop on Quality of Service*, Charleston, SC, Jul. 2009, pp. 1–9.

[8] Cloud Security Alliance, "Security Guidance for Critical Areas of Focus in Cloud Computing," 2012. [Online]. Available: https://cloudsecurityalliance.org/guidance/csaguide.v3.0.pdf

[9] H. Shacham and B. Waters, "Compact Proofs of Retrievability," in *International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology (Asiacrypt)*, Melbourne, Dec. 2008, pp. 90–107.

[10] A. Juels and B. S. Kaliski, "PORs: Proofs of Retrievability for Large Files," in *ACM Conference on Computer and Communication Security (CCS)*, Alexandria, VA, Oct. 2007, pp. 584–597.

[11] C. Wang, Q. Wang, K. Ren, and W. Lou, "Privacy-Preserving Public Auditing for Data Storage Security in Cloud Computing," in *IEEE International Conference on Computer Communications (INFOCOM)*, San Diego, CA, Mar. 2010, pp. 1–9.

[12] G. Ateniese, R. Di Pietro, L. V. Mancini, and G. Tsudik, "Scalable and efficient provable data possession," in *International Conference on Security and Privacy in Communication Networks (SecureComm)*, Istanbul, Sep. 2008, pp. 1–10.

[13] K. D. Bowers, A. Juels, and A. Oprea, "Proofs of Retrievability : Theory and Implementation," in *ACM Workshop on Cloud Computing Security (CCSW)*, Chicago, IL, Nov. 2009, pp. 43–54.

[14] C. Erway, A. Kupcu, C. Papamanthou, and R. Tamassia, "Dynamic Provable Data Possession," in *ACM Conference on Computer and Communication Security (CCS)*, Chicago, IL, Nov. 2009, pp. 213–222.

[15] Q. Wang, C. Wang, J. Li, K. Ren, and W. Lou, "Enabling Public Verifiability and Data Dynamics for Storage Security in Cloud Computing," in *European Conference on Research in Computer Security (ESORICS)*, Saint Malo, Sep. 2009, pp. 355–370.

[16] S. Yu, C. Wang, K. Ren, and W. Lou, "Achieving Secure, Scalable, and Fine-grained Data Access Control in Cloud Computing," in *IEEE International Conference on Computer Communications (INFOCOM)*, San Diego, CA, Mar. 2010, pp. 1–9.

[17] C. Wang, Q. Wang, K. Ren, N. Cao, and W. Lou, "Towards Secure and Dependable Storage Services in Cloud Computing," *(to appear) IEEE Transactions on Services Computing*, 2011.

[18] B. Chen, R. Curtmola, G. Ateniese, and R. Burns, "Remote Data Checking for Network Coding-based Distributed Storage Systems," in *ACM Workshop on Cloud Computing Security (CCSW)*, Chicago, IL, Oct. 2010, pp. 31–42.

[19] T. K. Dikaliotis, A. G. Dimakis, and T. Ho, "Security in Distributed Storage Systems by Communicating a Logarithmic Number of Bits," in *IEEE International Symposium on Information Theory (ISIT)*, Austin, TX, Jun. 2010, pp. 1948–1952.

[20] A. Le and A. Markopoulou, "Locating Byzantine Attackers in Intra-Session Network Coding using SpaceMac," in *IEEE International Symposium on Network Coding (NetCod)*, Toronto, Jun. 2010, pp. 1–6.

[21] ——, "On Detecting Pollution Attacks in Inter-Session Network Coding," in *IEEE International Conference on Computer Communications (INFOCOM)*, Orlando, FL, Mar. 2012, pp. 343–351.

[22] I. Damgard, "Towards Practical Public Key Systems Secure Against Chosen Ciphertext Attacks," Santa Barbara, CA, pp. 445–456, 1992.

[23] A. Dimakis, "Distributed Storage Wiki," 2012. [Online]. Available: http://csi.usc.edu/~dimakis/StorageWiki

[24] Y. Hu, H. C. H. Chen, P. P. C. Lee, and Y. Tang, "NCCloud: Applying Network Coding for the Storage Repair in a Cloud-of-Clouds," in *USENIX Conference on File and Storage Technologies (FAST)*, San Jose, CA, Feb. 2012, pp. 265–272.

[25] C. Gkantsidis and P. Rodriguez, "Cooperative security for network coding file distribution," in *IEEE Interna-*

tional Conference on Computer Communications (INFOCOM), Barcelona, Apr. 2006, pp. 1–13.

[26] S. Agrawal and D. Boneh, "Homomorphic MACs: MAC-based integrity for network coding," in *Applied Cryptography and Network Security (ACNS)*, Paris, Jun. 2009, pp. 292–305.

[27] Y. Li, H. Yao, M. Chen, S. Jaggi, and A. Rosen, "RIPPLE Authentication for Network Coding," in *IEEE International Conference on Computer Communications (INFOCOM)*, San Diego, CA, Mar. 2010, pp. 1–9.

[28] D. Boneh, D. Freeman, J. Katz, and B. Waters, "Signing a Linear Subspace : Signature Schemes for Network Coding," in *Public Key Cryptography (PKC)*, Irvine, CA, Mar. 2009, pp. 68–87.

[29] P. Zhang, Y. Jiang, C. Lin, H. Yao, A. Wasef, and X. S. Shen, "Padding for Orthogonality : Efficient Subspace Authentication for Network Coding," in *IEEE International Conference on Computer Communications (INFOCOM)*, Shanghai, Apr. 2011, pp. 1026–1034.

[30] A. Le and A. Markopoulou, "TESLA-Based Defense Against Pollution Attacks in P2P Systems with Network Coding," in *IEEE International Symposium on Network Coding (NetCod)*, Beijing, Jul. 2011, pp. 1–7.

[31] ——, "Cooperative Defense Against Pollution Attacks in Network Coding Using SpaceMac," *(to appear) IEEE Journal on Selected Areas in Communications*, 2011.

[32] S. Pawar, S. E. Rouayheb, and K. Ramchandran, "On Secure Distributed Data Storage Under Repair Dynamics," in *IEEE International Symposium on Information Theory (ISIT)*, Austin, TX, Jun. 2010, pp. 2543–2547.

[33] S. E. Rouayheb, V. Prabhakaran, and K. Ramchandran, "Secure Distributive Storage of Decentralized Source Data: Can Interaction Help?" in *IEEE International Symposium on Information Theory (ISIT)*, Austin, TX, Jun. 2010, pp. 1953–1957.

[34] S. Pawar, S. E. Rouayheb, and K. Ramchandran, "Securing Dynamic Distributed Storage Systems from Malicious Nodes," in *IEEE International Symposium on Information Theory (ISIT)*, Saint Petersburg, Jul. 2011, pp. 1452–1456.

[35] L. Buttyan, L. Czap, and I. Vajda, "Pollution Attack Defense for Coding Based Sensor Storage," in *IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing (SUTC)*, Newport Beach, CA, Jun. 2010, pp. 66–73.

[36] A. Le and A. Markopoulou, "NC-Audit: Auditing for Network Coding Storage," in *(accepted to) IEEE International Symposium on Network Coding (NetCod)*, Cambridge, MA, Jun. 2012.

[37] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*. Chapman & Hall/CRC Press, 2007.

[38] S. Agrawal, D. Boneh, X. Boyen, and D. Freeman, "Preventing Pollution Attacks in Multi-Source Network Coding," in *Public Key Cryptography (PKC)*, Paris, May 2010, pp. 161–176.