UC Irvine

UC Irvine Previously Published Works

Title

Auditing for Distributed Storage Systems

Permalink

https://escholarship.org/uc/item/9dr7x848

Journal

IEEE/ACM Transactions on Networking, 24(4)

ISSN

1063-6692

Authors

Le, Anh
Markopoulou, Athina
Dimakis, Alexandros G

Publication Date

2016-08-01

DOI

10.1109/tnet.2015.2450761

Copyright Information

This work is made available under the terms of a Creative Commons Attribution License, availalbe at https://creativecommons.org/licenses/by/4.0/

Peer reviewed

eScholarship.org Powered by the California Digital Library
University of California

Auditing for Distributed Storage Systems

Anh Le, *Member, IEEE* Athina Markopoulou, *Senior Member, IEEE*
and Alexandros G. Dimakis, *Member, IEEE*

arXiv:1203.1730v5 [cs.CR] 12 May 2014

Abstract

Distributed storage codes have recently received a lot of attention in the community. Independently, another body of work has proposed integrity checking schemes for cloud storage, none of which, however, is customized for coding-based storage or can efficiently support repair. In this work, we bridge the gap between these two currently disconnected bodies of work. We propose NC-Audit, a novel cryptography-based remote data integrity checking scheme, designed specifically for network coding-based distributed storage systems. NC-Audit combines, for the first time, the following desired properties: (i) efficient checking of data integrity, (ii) efficient support for repairing failed nodes, and (iii) 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 message authentication code (MAC) scheme for network coding, and NCrypt, a novel chosen-plaintext attack (CPA) secure encryption scheme that preserves the correctness of SpaceMac. Our evaluation of NC-Audit based on a real Java implementation shows that the proposed scheme has significantly lower overhead compared to the state-of-the-art schemes for both auditing and repairing of failed nodes.

Index Terms

Network Coding, Distributed Storage, Auditing, Integrity, Encryption, Security.

I. Introduction

Traditional distributed storage architectures provide reliability through block replication, whose major disadvantage is the large storage overhead. As the amount of stored data is growing faster than hardware infrastructure, this becomes a major cost bottleneck. In contrast, coding techniques achieve higher data reliability with considerably smaller storage overhead [1]. For that reason,

A. Le and A. Markopoulou are with UC Irvine. Emails: {anh.le, athina}@uci.edu

A. G. Dimakis is with UT Austin. Email: dimakis@austin.utexas.edu

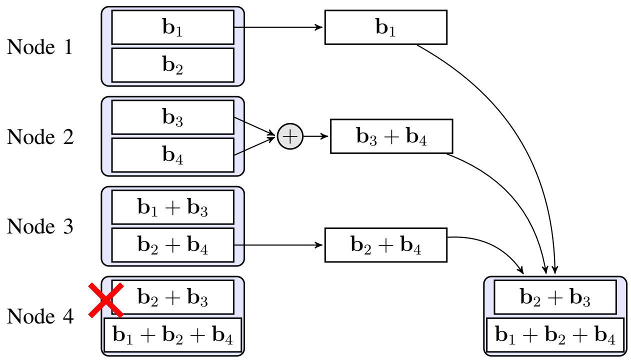
Node 1
$\mathbf{b}_1$
$\mathbf{b}_2$
$\mathbf{b}_1$
Node 2
$\mathbf{b}_3$
$\mathbf{b}_4$
$+$
$\mathbf{b}_3+\mathbf{b}_4$
Node 3
$\mathbf{b}_1+\mathbf{b}_3$
$\mathbf{b}_2+\mathbf{b}_4$
$\mathbf{b}_2+\mathbf{b}_4$
Node 4
$\mathbf{b}_2+\mathbf{b}_3$
$\mathbf{b}_1+\mathbf{b}_2+\mathbf{b}_4$
$\mathbf{b}_2+\mathbf{b}_3$
$\mathbf{b}_1+\mathbf{b}_2+\mathbf{b}_4$

Fig. 1. Repairing a failed node [6]: 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, where 4 is the amount of blocks needed to reconstruct the whole data.

coding techniques are under investigation for different distributed storage systems. Specifically, novel storage codes are currently being deployed in production cloud storage systems, such as Windows Azure [2], analytics clusters (*e.g.*, Facebook Analytics Hadoop clusters [3]), archival storage systems, and peer-to-peer storage systems like Cleversafe and Wuala [4], [5].

Distributed storage codes operate by splitting files into blocks and creating additional parity blocks that provide fault tolerance. If the original file consists of K blocks, an (N, K) maximum distance separable (MDS) code is typically used to produce N blocks to be stored individually on N storage nodes, thus tolerating up to $(N - K)$ node failures. A well-known problem of classical erasure codes, like Reed-Solomon, is the so-called repair problem: when a single node fails, typically one block is lost from the file; however, the reconstruction of that single block requires reading and transferring K blocks from other nodes.

Novel storage codes that use network coding (NC) were recently developed to reduce this *repair bandwidth*. These distributed storage codes require significantly less than K blocks to repair a single node failure and rely on network coding to perform in-network processing [6], [7]. Key ingredients of NC-based distributed storage codes include (i) storing *coded blocks*, *i.e.*, linear combinations of original blocks that form the original data, and (ii) *block mixing* when repairing. An example is shown in Fig. 1. The repair bandwidth, however, is only one aspect of cloud storage.

Another practical aspect of cloud storage, besides the repair bandwidth, is data integrity checking. Integrity checking is extremely important for distributed storage systems, especially when data is stored with untrusted cloud providers. Data can be lost or corrupted for various reasons while users may remain completely unaware of for long periods of time. For example, storage errors, such as torn writes [8] and latent errors [9], may damage data in a way that remains undetected. Cloud storage providers may also have incentives to misbehave, *e.g.*, misreport data loss incidents in order to maintain their reputation [10]–[12]. This problem is further exacerbated in systems that use coding because corrupted data can propagate to multiple nodes during repair re-encoding [13]. Therefore, it is important for the user to be able to audit the integrity of the data stored on the cloud.

Another complication is that frequent integrity checking of large data sets may be out of the ability or budget of users with limited resources [12], [14]. As a result, users often resort to a third party to perform audits on their behalf [10], [12], [15], [16]. In this latter case, it is important that the auditing protocols are privacy-preserving, *i.e.*, do not leak information to the third party [12], [17]. Indeed, users can leverage data encryption to protect their data before outsourcing it [16]. However, data encryption should be complementary and orthogonal to integrity checking protocols. In other words, the auditing protocol should not introduce new vulnerabilities of unauthorized data leakage. Furthermore, the users may want to outsource unencrypted instead of encrypted data to support more efficient and complex computations.

As a result, auditing for distributed systems that use modern NC-based storage codes is an important emerging problem. Despite the rich literature on auditing protocols for general distributed and cloud storage [10]–[12], [15]–[20], [22]–[25], there have been very few auditing protocols for NC-based distributed storage systems [13], [26]. These protocols, however, are generic in the sense that they do not specifically exploit coding properties for efficient integrity checking [13]. Moreover, they do not prevent data leakage [13], [26]. Most importantly, they do not efficiently support repair, which is the main advantage of NC-based storage systems when compared to other storage systems.

In this work, we propose a symmetric key-based cryptographic protocol, called NC-Audit, to check for the integrity of data stored on an 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 desired properties:

(i) **Efficient Integrity Checking**: The integrity check incurs a small bandwidth and computational overhead (on the order of 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**: The repair of failed nodes require negligible bandwidth (no data download) as well as computation 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 ($< 1\%$) and computational overhead (on the order of milliseconds).

We would like to emphasize that, independently of (iii), properties (i) and (ii) together are already useful to users who could and prefer to audit the data themselves. NC-Audit is the first protocol that possesses (i) and (ii) at the same time. NC-Audit achieves these properties by fully exploiting network coding in its design. The main novelty of NC-Audit come from a careful combination of SpaceMac – a homomorphic message authentication code (MAC) that was previously specifically designed for network coding [27], [28], and NCrypt – a novel chosen-plaintext attack (CPA) secure encryption scheme that we custom designed, in this work, to operate in synergy with and preserve the correctness of SpaceMac.

We implemented NC-Audit in Java, utilizing our previous implementation of SpaceMac [28]. Our evaluation of NC-Audit shows that it has very low computational overhead. In particular, when performing an audit, both the storage node and the auditor only need to spend a few milliseconds. Furthermore, the auditor's overhead is much less than that of the state-of-the-art approach for NC-based storage systems [13], which is on the order of seconds.

The rest of the paper is organized as follows. In Section II, we discuss related work. In Section III, we formulate the problem and describe the threat model. In Section IV, we describe the auditing framework and the key building blocks of NC-Audit, namely SpaceMac and NCrypt, before presenting NC-Audit itself. In Section V, we show how NC-Audit efficiently supports repair. In Section VI, we analyze the security of NC-Audit. In Section VII, we evaluate its storage, bandwidth, and computational efficiency. In Section VIII, we conclude the paper.

II. Related Work

A. Integrity Checking for Remote Data

There has been a rich body of work on integrity checking for remote data [10]–[12], [15]–[19], [22]–[25], commonly known as *Proof of Retrievability* and *Proof of Data Possession*.

Proof of Retrievability (POR). In [16], 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 there 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. In contrast, NC-Audit does not use sentinels and supports unlimited number of queries.

In [15], Shacham and Waters proposed two POR schemes with full proofs of security and extract-ability. The first one, built on Boneh-Lynn-Shacham (BLS) signatures, provides public verifiability. The second one, built on pseudorandom functions (PRFs), provides private verifiability. Recently, Bowers *et al.* [20] proposed HAIL, an improvement of existing POR schemes that allows for performing data integrity checking with multiple servers against stronger, mobile adversaries.

These schemes [15], [20] exploit homomorphic properties to aggregate authenticator values to improve the audit efficiency. NC-Audit also exploits homomorphic properties (of SpaceMac) and provides private verifiability. In terms of extract-ability, NC-Audit is different from existing approaches, *e.g.*, [15], in that NC-Audit exploits the inherent embedded coding coefficients in the stored blocks to perform the extraction. Meanwhile, [15] relies on additional erasure codes (pre-applied to the data) for the extraction.

Proof of Data Possession (PDP). The notion of PDP was introduced by Ateniese *et al.* [10]. The PDP scheme in [10] uses homomorphic RSA signatures to generate verification tags. The data possession guarantee provided by this scheme is under the RSA and KEA1 assumptions in the random oracle model. Earlier in [29], Schwarz and Miller proposed using a combination of both erasure-correcting coding and algebraic signatures (homomorphic hashes) to perform integrity checking for remote data. As discussed in [15], the notion of PDP is considered to be 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. Integrity checking for groups with efficient user revocation was recently introduced in [21]. We will show that NC-Audit provides the stronger data possession checking with data extraction as in POR (Section VI-A).

Data Modification. In [18], Ateniese *et al.* proposed a symmetric-key based checking scheme that supports data modification. This scheme is built on regular PRFs, hash functions, and encryptions. It provides private verifiability and supports a limited number of queries. In [22], 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. In [23], Wang *et al.* proposed a public auditing scheme that uses a combination of the BLS-based scheme in [15] and Merkle Hash Tree (MHT).

In practice, most current deployments of distributed storage codes [2], [3] initially set all files to replication mode. When certain files become *cold* (*i.e.*, rarely accessed and modified) the replicated blocks are deleted and corresponding parity blocks are created. This dynamic switching of files from replication to coding allows distributed storage systems to benefit from the high performance of replication for *hot* files and the storage benefits of coding for *cold* files. Interestingly, in most analytics clusters and cloud storage systems, the vast majority of data seem to be *cold* [2], [3]. Therefore, we do not expect data modification to be a critical operation for encoded data. NC-Audit provides some preliminary support for data modification, and the details can be found in the Appendix.

Privacy Preserving. In [11], Shah *et al.* proposed an auditing protocol that is privacy preserving. This protocol first encrypts the data and then sends 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 [17], Wang *et al.* proposed a privacy preserving auditing protocol that has public verifiability. This protocol can be considered an extension of the BLS-based protocol in [15]. 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 IV-E and VI-B). Different from [17], NC-Audit relies on symmetric-key cryptographic primitives instead of public-key ones, and thus it provides private instead of public auditing.

Finally, we stress that none of the schemes described above was customized for NC-based storage. In particular, they do not provide efficient support for node repair. NC-Audit was designed to achieve all the above properties: providing proof of retrievability and privacy-preserving auditing while efficiently supporting node repair.

B. 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.* [7]. In particular, in [7], 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. A survey on recent advances in NC-based storage system can be found at [6]. A wiki on NC-based storage cloud is maintained at [31]. NC-Audit is designed to *fully support regenerating codes*.

An NC-based distributed file system (NCFS) is proposed in [32]. One of the first implementations of NC-based storage cloud is NCCloud by Hu *et al.* [33]. 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 on top of Windows Azure Storage.

Integrity Checking Schemes for NC-Based Storage Systems. There have been only a few number of work that provide remote data checking for NC-based storage. In [26], 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 [13]. In this work, Chen *et al.* adopted the symmetric-key based scheme that Shacham and Waters proposed for regular cloud storage [15] with minor modification. In particular, based on the symmetric-key based scheme in [15], the scheme in [13] 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 [13]. NC-Audit overcomes this attack by requiring the user/auditor to store the coding coefficients, which is also needed for the repair process and only occupies a negligible amount of storage (see Section VII-A).

What really sets NC-Audit apart from [13] is that NC-Audit fully exploits network coding for integrity checking. In particular, the scheme proposed in [13] relies on two independent logical representation of file blocks for two different purposes: data possession checking and network coding operation. Because of this, during the repair process, the user has to download blocks from the remaining healthy nodes to compute the integrity checking data for the new coded blocks (to be stored at the recovery node). This approach puts heavy bandwidth and computational overhead on the user. In contrast, NC-Audit uses a *single* representation for both purposes and thereby achieving integrity checking while eliminating the heavy user's bandwidth and computational overhead. Details of how NC-Audit support efficient repair are provided in Section V. Furthermore, the scheme in [13] does not support privacy-preserving auditing while NC-Audit does. We provide detailed performance comparison between NC-Audit and [13] in Section VII.

Finally, a recent work by Cao *et al.* [34] proposed an LT codes-based storage system with an integrity checking and an exact repair schemes; however, it neither supports functional repair [7] (discussed in Section V) nor privacy-preserving auditing.

Other Security Issues. Other security problems for NC-based storage include protecting the privacy and integrity of the blocks while repairing. The work in [42] and [43] prevents eavesdroppers from accessing/decoding all the data. In [42], 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. [43] analyzes the effects of interaction between the storage nodes

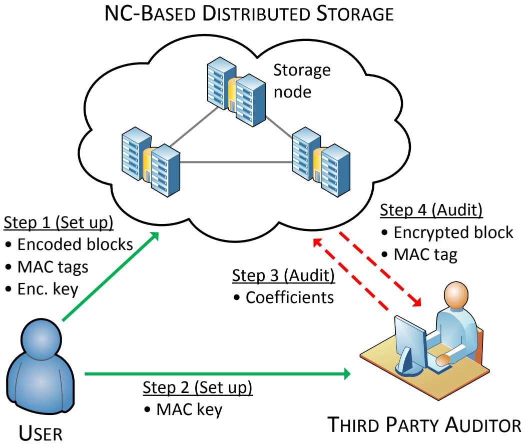

Fig. 2. Parties and Steps Involved in NC-Audit.

on the amount of data revealed to the eavesdroppers. The work in [44] provides upper bounds on the maximum amount of information that can be stored safely when there are malicious nodes.

In [44] and [45], the authors provide protection against pollution attacks during the repair. In [45], Buttyan *et al.* provide a lightweight, pollution-resilient decoding algorithm that is capable of finding adversarial blocks. The scheme in [13] also protects the repair phase against pollution attacks, *i.e.*, preventing remaining nodes from sending corrupted data to the new (recovering) node. 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 [35]–[41].

C. This Work in Perspective

A preliminary version of this work has appeared in NetCod 2012 [46]. In this paper, we provide the following revisions and extensions of the previous version: We revise and provide complete proofs of all lemmas and theorems; we described in detail a repair process; we discuss and compare our storage overhead to prior work [13], [17], [23]; finally, we provide a comprehensive discussion of related literature.

III. Problem Formulation

A. System Model and Operations

Fig. 2 illustrates an overview of NC-Audit. 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 his/her data on the storage nodes. The user resorts to a TPA to check for the integrity of the data stored at each node; at the same time, he/she does not want the TPA to learn about the outsourced data. We assume that *the user is responsible for repairing of a failed node.* The user here acts as a proxy that manages the storage nodes as in the case of NCCloud [33]. Our work is also applicable to scenarios where there is a cloud service provider, who is independent from the user and acts as the proxy.

The user follows the following basic steps to store his/her data on the storage cloud. We adopt the notation used in [40]. 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:

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

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 [26].

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, P be the number of healthy nodes that need to send the (encoded) repair blocks, and Q be the number of repair blocks each healthy node needs to send to the new node. In the example given in Fig. 1, $m = 4$, $N = 4$, $M = 2$, $P = 3$, and $Q = 1$.

B. Threat Model

We adopt the threat model considered in [17] and [24]. In particular, we consider semi-trusted storage nodes that behave properly and do not deviate from the prescribed auditing protocol. However, for their own benefits, they may deliberately delete rarely accessed, archival user's data to reduce operational cost; 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.

We assume that the TPA, who is in the business of auditing, is reliable and independent. We assume that the TPA does *not collude* with the storage node during the auditing process to hide data corruption. This is a standard assumption when relying on a TPA for integrity checking [17], [21], [23]. 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, as in [17]. In order words, the auditing protocol should not introduce a data leakage vulnerability. Similar to standard applications of cryptographic protocols, 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.

IV. Auditing Scheme

A. *Definitions and Auditing Framework*

We follow the literature of integrity checking of remote data [10], [15]–[17], [19] and adapt the proposed framework to our privacy-preserving auditing system. In particular, we consider an auditing scheme which consists of four algorithms:

- $\mathsf{KeyGen}(1^\lambda) \to (k_v, k_e)$ is a 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_v used to generate *verification* metadata, and k_e used to *encrypt* the possession proof.
- $\mathsf{TagGen}(\mathbf{e}, k_v) \to t$ is an algorithm run by the user to generate the verification metadata. It takes as input a coded block, $\mathbf{e}$, a private key, k_v, and outputs a verification tag of $\mathbf{e}$, t.
- $\mathsf{GenProof}(k_e, (\mathbf{e}_1, \cdots, \mathbf{e}_M), (t_{\mathbf{e}_1}, \cdots, t_{\mathbf{e}_M}), \mathsf{chal}) \to V$ is run by the storage node to generate a proof of possession. It takes as input a private key, k_e; 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, which includes block indices and coding coefficients. It outputs a proof of possession, V, for the coded blocks determined by chal.
- $\mathsf{VerifyProof}(k_v, \mathsf{chal}, V) \to \{1, 0\}$ is run by the TPA in order to validate a proof of possession. It takes as inputs a private key, k_v, 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 previously described in Section III-A. 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.

B. 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 [10]. 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_v, and a standard MAC scheme, *e.g.*, HMAC. The user then uploads both the tags and the coded blocks to the storage node and sends k_v 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_v 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 since the computation and bandwidth overhead increases linearly in the number of checked blocks.
- It does not efficiently support *node repair* [6], [7]: It requires the user to download all the blocks necessary to compute the new (recovering) blocks. The user then computes verification tags for all the new blocks, essentially re-setting up the storage node.

- It violates privacy because the TPA learns about the blocks. A straightforward way to provide privacy is to encrypt the response block using a standard encryption scheme, *e.g.*, AES. However, in this case, the TPA will not be able to verify the integrity of the original block because the provided tag is not computed on the encrypted block but on the original block.

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

In detail, we adopt SpaceMac, a homomorphic MAC scheme that we previously designed specifically for network coding [27], [41]. 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 a 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 preserves the correctness of SpaceMac: A response block, even when encrypted, can be used by the TPA for the integrity check. We stress that it is not possible to use other standard encryption schemes, such as AES, in place of NCrypt, because they will break the SpaceMac integrity verification. The reason is that in general, a MAC tag computed on a data block can only be used to verify the integrity of the block upon the reception of the tag and the data block, but it cannot be used when the encrypted data block is received instead of the original block.

Formally, let $(\mathsf{Enc}, \mathsf{Dec})$ denote a symmetric-key encryption scheme and $(\mathsf{Mac}, \mathsf{Verify})$ denote a MAC scheme. Let $\mathbf{e}$ be an (encoded) data block, and k_e and k_v be the keys for the encryption and MAC schemes. Let $\mathbf{c} = \mathsf{Enc}(k_e, \mathbf{e})$ and $t = \mathsf{Mac}(k_v, \mathbf{e})$. The encryption and MAC schemes are compatible with each other when $\mathsf{Verify}(k_v, \mathbf{c}, t)$ outputs 1 if and only if $\mathbf{c} = \mathsf{Enc}(k_e, \mathbf{e})$ and outputs 0 otherwise.

The main novelty of NCrypt lies in its compatibility with SpaceMac: It is carefully designed to maintains both the correctness of SpaceMac (Theorem 3) as well as the security of SpaceMac (Theorem 4). NCrypt employs the random linear combination technique of network coding and is semantically secure under a chosen-plaintext attack (CPA-secure). Next, we describe how we use SpaceMac and NCrypt in detail.

C. The Homomorphic MAC: SpaceMac

In prior work, we originally designed SpaceMac and used it to combat pollution attacks in network coding [27], [28], [40], [41]. SpaceMac was inspired by and an improvement of another homomorphic MAC scheme, HomMac, proposed by Agrawal and Boneh [36]. The novelty of SpaceMac and a detailed comparison between the two schemes can be found in [27], [41]. Here, we adopt SpaceMac to support the aggregation of file blocks and tags to allow for efficient auditing (similar to [15], [20]). Furthermore, as we show in Section V, SpaceMac also facilitates efficient node repairs.

Definition. A (q, n, m) homomorphic MAC scheme is defined by three probabilistic, polynomial-time algorithms: Mac, Combine, and Verify. The Mac algorithm generates a tag for a given block; the Combine algorithm computes a tag for a linear combination of some given blocks; and the Verify algorithm verifies whether a tag is a valid tag of a given block.

- $\mathsf{Mac}(k, \text{id}, \mathbf{e})$:
  - Input: A secret key, k, the identifier, id, of the file, and a source block or encoded block, $\mathbf{e} \in \mathbb{F}_q^{n+m}$.
  - Output: Tag t for $\mathbf{e}$.
- $\mathsf{Combine}((\mathbf{e}_1, t_1, \alpha_1), \cdots, (\mathbf{e}_\ell, t_\ell, \alpha_\ell))$:
  - Input: ℓ blocks, $\mathbf{e}_1, \cdots, \mathbf{e}_\ell$, their tags, $t_1, \cdots, t_\ell$, under key k, and their coefficients, $\alpha_1, \cdots, \alpha_\ell \in \mathbb{F}_q$.
  - Output: Tag t for block $\mathbf{e} \overset{\text{def}}{=} \sum_{i=1}^{\ell} \alpha_i \, \mathbf{e}_i$.
- $\mathsf{Verify}(k, \text{id}, \mathbf{e}, t)$:
  - Input: A secret key, k, the identifier, id, of the file, a block, $\mathbf{e} \in \mathbb{F}_q^{n+m}$, and its tag, t.
  - Output: 0 (reject) or 1 (accept).

Also, the scheme must satisfy the following correctness requirement:
Let $t = \mathsf{Combine}((\mathbf{e}_1, t_1, \alpha_1), \cdots, (\mathbf{e}_\ell, t_\ell, \alpha_\ell))$, then $\mathsf{Verify}\left(k, \text{id}, \sum_{i=1}^{\ell} \alpha_i \mathbf{e}_i, t\right) = 1$.

Note that the homomorphic property of the MAC scheme, or the existence of Combine, which does not exist in regular MAC schemes, such as HMAC, ensures that multiple blocks can be audit at the bandwidth and verification computation cost of a single block.

Construction. SpaceMac consists of a triplet of algorithms: Mac, Combine, and Verify. The construction of SpaceMac uses a pseudo-random function (PRF) $F_1 : \mathcal{K}_1 \times (\mathcal{I} \times [1, n+m]) \to \mathbb{F}_q$, where $\mathcal{K}_1$ is the PRF key domain and $\mathcal{I}$ is the file identifier domain.

- $\mathsf{Mac}(k, \mathsf{id}, \mathbf{e}) \to t$: 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:
  – $\mathbf{r} \leftarrow (F_1(k, \mathsf{id}, 1), \cdots, F_1(k, \mathsf{id}, n+m))$.
  – $t \leftarrow \mathbf{e} \cdot \mathbf{r} \in \mathbb{F}_q$.
- $\mathsf{Combine}((\mathbf{e}_1, t_1, \alpha_1), \cdots, (\mathbf{e}_\ell, t_\ell, \alpha_\ell)) \to t$: The tag $t \in \mathbb{F}_q$ of $\mathbf{e} \overset{\text{def}}{=} \sum_{i=1}^{\ell} \alpha_i \mathbf{e}_i \in \mathbb{F}_q^{n+m}$ is computed as follows:
  – $t \leftarrow \sum_{i=1}^{\ell} \alpha_i t_i \in \mathbb{F}_q$.
- $\mathsf{Verify}(k, \mathsf{id}, \mathbf{e}, t) \to \{0, 1\}$: 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 [27])**.** *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 [27] for the security game and proof of SpaceMac. We provide security proof of SpaceMac when used in NC-Audit in Section VI-A. If the user computes the verification tags for the source blocks using the Mac algorithm of SpaceMac, then the storage node can compute a valid MAC tag for any encoded block using the Combine algorithm. 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.

D. 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. To this end, we design a novel encryption scheme that is compatible with SpaceMac, called NCrypt. In particular, NCrypt will protect $n-2$ elements of the response block while still allowing SpaceMac integrity checking. The remaining 2 elements are random padded elements. These 2 elements are needed to guarantee the security of the schemes, as we will show in the construction and proofs of NCrypt and SpaceMac[1].

Let $\bar{\mathbf{x}}$ denote the vector formed by the first $n-2$ elements of a vector $\mathbf{x}$. The construction of NCrypt uses two PRFs: $F_2 : \mathcal{K}_2 \times ([1, n-1] \times [1, n-2]) \to \mathbb{F}_q$ and $F_3 : \mathcal{K}_2 \times (\{0,1\}^\lambda \times [1, n-1]) \to \mathbb{F}_q$, where $\mathcal{K}_2$ is a PRF key domain. NCrypt consists of three probabilistic, polynomial time algorithms:

- $\mathsf{Setup}(k, \bar{\mathbf{r}}) \to (p_1, \cdots, p_{n-1})$: This algorithm is run by the user to setup the encryption scheme. It takes as input a secret key k and a vector $\bar{\mathbf{r}} \neq \mathbf{0}, \bar{\mathbf{r}} \in \mathbb{F}_q^{n-2}$. It outputs $n-1$ elements in $\mathbb{F}_q$, which are called *auxiliary elements* and are used by the encryption. The details are as follows:
  – Compute $\bar{\mathbf{p}}_i \leftarrow (F_2(k, i, 1), \cdots, F_2(k, i, n-2)) \in \mathbb{F}_q^{n-2}$, for $i \in [1, n-1]$.
  – Compute $p_i \leftarrow \bar{\mathbf{r}} \cdot \bar{\mathbf{p}}_i \in \mathbb{F}_q$, for $i \in [1, n-1]$.
- $\mathsf{Enc}(k, \bar{\mathbf{e}}, (p_1, \cdots, p_{n-1})) \to \langle \bar{\mathbf{c}}, (r, p) \rangle$: This algorithm is run by the storage node to encrypt the $n-2$ first elements of the aggregated response block. It takes as input a secret key, k, vector formed by the first $n-2$ elements of the response block, $\bar{\mathbf{e}}$, and the auxiliary 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*: $\beta_i \leftarrow F_3(k, r, i) \in \mathbb{F}_q$, for $i \in [1, n-1]$.
  – Compute the *masking vector*: $\bar{\mathbf{m}} \leftarrow \sum_{i=1}^{n-1} \beta_i \bar{\mathbf{p}}_i \in \mathbb{F}_q^{n-2}$.
  – Compute $\bar{\mathbf{c}} \leftarrow \bar{\mathbf{e}} + \bar{\mathbf{m}} \in \mathbb{F}_q^{n-2}$.
  – 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 $\bar{\mathbf{m}} \in \mathsf{span}(\bar{\mathbf{p}}_\mathbf{1}, \cdots, \bar{\mathbf{p}}_{\mathbf{n-1}})$.
- $\mathsf{Dec}(k, \langle \bar{\mathbf{c}}, (r, p) \rangle) \to \bar{\mathbf{e}}$: This algorithm takes as input a secret key, k, and the cipher text,

[1] In particular, the 2 random padded elements is to control the number of equations in the system of equations Π_1 and Π_2 described in the proofs of Theorems 2 and 4, respectively. Intuitively, these 2 random elements are needed to compensate for the extra information learned by the adversary in NCrypt (the element p as part of the ciphertext) and in SpaceMac (the equations related to $\bar{\mathbf{r}}$).

$\langle \bar{\mathbf{c}}, (r, p) \rangle$. 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 $\bar{\mathbf{m}} \leftarrow \sum_{i=1}^{n-1} \beta_i \, \bar{\mathbf{p}}_i \in \mathbb{F}_q^{n-2}$.

– Compute $\bar{\mathbf{e}} \leftarrow \bar{\mathbf{c}} - \bar{\mathbf{m}} \in \mathbb{F}_q^{n-2}$.

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 $(n-2) \times \log_2 q$ 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 β_i's without knowing the secret key k.

The proof follows a textbook technique used to prove the security of Construction 3.24 in [47]. We follow the notation in [47]. 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. $\mathcal{A}$ outputs a pair of messages $\bar{\mathbf{e}}_0$ and $\bar{\mathbf{e}}_1$, both are in $\mathbb{F}_q^{n-2}$.
- A random bit $b \leftarrow \{0,1\}$ is chosen, and then a ciphertext $c \leftarrow \mathsf{Enc}(k, \bar{\mathbf{e}}_b, (p_1, \cdots, p_{n-1}))$ 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 $\mathrm{Adv}[\mathcal{B}, F_2]$ be the probability of an adversary $\mathcal{B}$ with similar runtime to $\mathcal{A}$ winning the PRF security game (can tell a pseudo-random function F_2 from a truly random function f_2). By the security of PRF, we have that $\mathrm{Adv}[\mathcal{B}, F_2]$ is negligible in λ and it can be shown that (details are provided in the proof of Construction 3.24 in [47])

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

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 Adv$[\mathcal{C}, F_3]$ be the probability of an adversary $\mathcal{C}$ with similar runtime to $\mathcal{A}$ winning the PRF security game. Similar to the above, by the security of PRF, we have that Adv$[\mathcal{B}, F_3]$ is negligible in λ and

$$\mathsf{Adv}[\mathcal{C}, F_3] = \left|\mathsf{Pr}[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\Pi_1} = 1] - \mathsf{Pr}[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\Pi_2} = 1]\right|. \tag{2}$$

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

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

Let r_c denote the random string used when generating the challenge ciphertext, which is of the form $\langle \bar{\mathbf{c}}, (r_c, p)\rangle$ (by encrypting $\bar{\mathbf{e}}_b$). There are two cases:

(a) r_c is never used by the oracle in the encryption algorithm to produce ciphertext to answer any of $\mathcal{A}$'s queries: In the following, we will show that each element of *any* plaintext $\bar{\mathbf{e}}$ is masked with a uniformly random value, thus the adversary will not be able to tell which message ($\bar{\mathbf{e}}_0$ or $\bar{\mathbf{e}}_1$) was encrypted, as in the case of one-time pad.

Parse $\bar{\mathbf{e}}$ as $(e^{(1)}, \cdots, e^{(n-2)})$, $\bar{\mathbf{m}}$ as $(m^{(1)}, \cdots, m^{(n-2)})$, and $\bar{\mathbf{p}}_i$ as $(p_i^{(1)}, \cdots, p_i^{(n-2)})$. From a ciphertext returned from an oracle query of $\bar{\mathbf{e}}$, the adversary can construct the following system of equations Π_1 by subtracting the query plaintext from the ciphertext:

$$(\Pi_1) \quad \begin{cases} \beta_1 \, p_1^{(1)} + \cdots + \beta_{n-1} \, p_{n-1}^{(1)} & = m^{(1)} \\ \cdots & \\ \beta_1 \, p_1^{(n-2)} + \cdots + \beta_{n-1} \, p_{n-1}^{(n-2)} & = m^{(n-2)} \\ \beta_1 p_1 + \cdots + \beta_{n-1} p_{n-1} & = p \end{cases}.$$

Note that $p_i^{(j)}$ are not all zeros w.h.p. since they are chosen uniformly at random from $\mathbb{F}_q$ by f_2. Let β_i be unknowns, $i \in [1, n-1]$. The above system of $n-1$ 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. Let s be the rank of the coefficient matrix. Now for any $w \in [1, n-2]$, assume that all $m^{(j)}, j \neq w, j \in [1, n-2]$, 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 $\mathbb{F}_q$ (as a truly random function f_3 is used in place of F_3). Thus, each element of the plaintext, $e^{(w)}$, is masked with a uniformly random value, $m^{(w)}$, independent of other masking elements $m^{j \neq w}, j \in [1, n-2]$. 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, $\langle \bar{\mathbf{c}}, (r, p) \rangle$, it learns the masking vector $\bar{\mathbf{m}}$ associated with r as $\bar{\mathbf{m}} = \bar{\mathbf{c}} - \bar{\mathbf{e}}$. Thus, by leveraging the corresponding $\bar{\mathbf{m}}$ of r_c, the adversary can tell if $\bar{\mathbf{e}}_0$ or $\bar{\mathbf{e}}_1$ was encrypted by actually decrypting the challenge response. 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) show that

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

where ϵ is a cryptographically negligible function in λ. This completes the proof. ■

E. The Privacy-Preserving Auditing Scheme: NC-Audit

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

Setup phase:

- The user divides the file into m blocks of size $n-2$ instead of n and pads to each block two random elements in $\mathbb{F}_q$. This is necessary as NCrypt encrypts only the first $n-2$ 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 verification key, k_v, and encryption key, k_e:
  – $\mathsf{KeyGen}(1^{\lambda}) \to (k_e, k_v)$: $k_e \xleftarrow{R} \{0,1\}^{\lambda}, k_v \xleftarrow{R} \{0,1\}^{\lambda}$.
- The user then setups the encryption scheme by computing the auxiliary elements, $p_1, \cdots, p_{n-1}$:
  – $\bar{\mathbf{r}} \leftarrow (F_1(k_v, 1), \cdots, F_1(k_v, n-2))$.
  – $(p_1, \cdots, p_{n-1}) \leftarrow \mathsf{Setup}(k_e, \bar{\mathbf{r}})$.
- Afterward, the user computes a tag for each source block $\mathbf{b}_i$ using the Mac algorithm of SpaceMac:
  – $t_{\mathbf{b}_i} = \mathsf{Mac}(k_v, \mathbf{b}_i)$.
- 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:
  – $\mathsf{TagGen}(\mathbf{e}, k_v) \to t_{\mathbf{e}} = \sum_{i=1}^{m} \alpha_i \, t_{\mathbf{b}_i}$.

- 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 auxiliary elements, $p_1, \cdots, p_{n-1}$, and the encryption key, k_e, 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_v, to the TPA. We assume that the user uses private and authentic channels to send k_v and k_e[2]. The user then keeps the coding coefficients and the keys but delete all other data.

Note that maintaining coding coefficients is necessary for the repair process and is an inherent characteristic of NC storage systems. The overhead of storing the coefficients is negligible compared to the outsource data and could be constant for practical purposes (see Section VII-A). If the user outsources the management of the nodes to a third party, such as a proxy as in NCCloud [33], then he/she does not need to store the coding coefficients. However, in this case, the proxy must be trusted.

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: $\alpha_i \overset{R}{\leftarrow} \mathbb{F}_q, i \in \mathcal{I}$. 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 $(\bar{\mathbf{e}}, e^{(n-1)}, e^{(n)})$.

  – Compute the aggregated tag: $t = \sum_{i=\in\mathcal{I}} \alpha_i \, t_{\mathbf{e}_i}$.

  – Encrypt the response block: $\langle \bar{\mathbf{c}}, (r, p)\rangle \leftarrow \mathsf{Enc}(k_e, \bar{\mathbf{e}}, (p_1, \cdots, p_{n-1}))$.

  The node then sends $V = (\langle \bar{\mathbf{c}}, (r, p)\rangle, e^{(n-1)}, e^{(n)}, t)$ 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 $\mathbf{c} = (\bar{\mathbf{c}} \,|\, e^{(n-1)} \,|\, e^{(n)} \,|\, \mathsf{aug}(\mathbf{e}))$, where "|" denotes augmentation.

  Return result of $\mathsf{Verify}(k_v, \mathbf{c}, t + p)$.

[2] Exchanging secret keys, in particular, and establishing secure and authentic channels, in general, could be done with the support of a public key infrastructure (PKI). This is an important, well studied problem in the cryptography community and is orthogonal to this work.

Correctness. The correctness of NC-Audit, *i.e.*, if the file is correct then the algorithm will accept the proof, is guaranteed by the following Lemma 3. And its security, *i.e.*, if there is corruption then the algorithm will reject the proof, is proved in Section VI.

Lemma 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 $\mathbf{r} = (F_1(k_v, 1), \cdots, F_1(k_v, n+m))$. Note that

$$\begin{aligned}\mathbf{c} &= (\bar{\mathbf{c}} \,|\, e^{(n-1)} \,|\, e^{(n)} \,|\, \mathsf{aug}(\mathbf{e})) \\ &= ((\bar{\mathbf{e}} + \bar{\mathbf{m}}) \,|\, e^{(n-1)} \,|\, e^{(n)} \,|\, \mathsf{aug}(\mathbf{e})) = \mathbf{e} + (\bar{\mathbf{m}} \,|\, 0, \cdots, 0)\,.\end{aligned}$$

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. ■

V. 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 that still preserve the same level of reliability (*functional repair*) [6]. In the example given in Fig. 1, the user instructs the first three storage nodes to send coded blocks to exactly repair the fourth node.

Formally, for each healthy node, $N_i, i = 1, \cdots, P$, recall that it needs to send Q encoded repair blocks to the new node. Let $(\mathbf{e}_{i,1}, \cdots, \mathbf{e}_{i,M})$ be the encoded blocks currently stored on N_i. For $j = 1, \cdots, Q$, the user sends a set of *repair coding coefficients* $(\gamma_{i,j,1}, \cdots, \gamma_{i,j,M})$ to N_i. This node then uses these coefficients to compute the repair blocks, $\mathbf{g}_{i,j} = \sum_{k=1}^{M} \gamma_{i,j,k} \, \mathbf{e}_{i,k}$, to send to the new node. The new node will receive $P \times Q$ repair blocks, $\mathbf{g}_{i,j}$, from the healthy nodes. It uses them to reconstruct the encoded blocks, $\mathbf{h}_1, \cdots, \mathbf{h}_M$, that it needs to store. For $k = 1, \cdots, M$, the user sends a set of $P \times Q$ *reconstruction coding coefficients*, $(\theta_{i,j,k}, \cdots \theta_{P,Q,k})$, to the new node to instruct its reconstruction. The new node then reconstructs $\mathbf{h}_k = \sum_{i=1}^{P} \sum_{j=1}^{Q} \theta_{i,j,k}\, \mathbf{g}_{i,j}$. Note that the coding coefficients γ's and θ's are dependent on the repairing scheme.

Using NC-Audit, the verification tags of the newly constructed blocks, $\mathbf{h}_k$, 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 repair blocks, $\mathbf{g}_{i,j}$, that they send to the new node, where the tags of $\mathbf{g}_{i,j}$ can be computed using the Combine algorithm of SpaceMac on the tags of $\mathbf{e}_{i,k}$. The new node then can also use Combine on the tags of $\mathbf{g}_{i,j}$ to generate tags of $\mathbf{h}_k$. Finally, the user sends the coding coefficients of the coded blocks at the newly constructed node, $\mathsf{aug}(\mathbf{h_k})$ (dependent on the repair scheme), to the TPA so that it can audit this new node.

Consequently, with NC-Audit, there is negligible cost to the user when repairing a failed node, in terms of both bandwidth and computation of verification metadata. In particular, the user does not need to download data, *i.e.*, $\mathbf{e}_{i,k}$, and the user also does not need to compute the tags, *i.e.*, runs Mac on $\mathbf{h}_k$. This stands in stark contrast with the prior integrity checking scheme for NC-based storage [13], which requires the user to download many data blocks (equal to the repair bandwidth) and compute security metadata for the newly coded blocks him/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* [13] (discussed in Section II-B); otherwise, it will not pass the audit because the SpaceMac tags are computed on both the data and coefficients. Here 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, including our own, which explicitly combats pollution attacks [28], [35]–[38], [40], [41], [45], [48].

VI. Security Analysis

A. Data Possession Guarantee

When using SpaceMac in NC-Audit, some information about the vector $\mathbf{r}$ in the SpaceMac construction is 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, $\mathsf{SpaceMac}$ is still a secure homomorphic MAC, *i.e.*, any corruption will be detected w.h.p.

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, $p_1, \cdots, p_{n-1}$, such that $\bar{\mathbf{p}}_i \cdot \bar{\mathbf{r}} = p_i$, where $\mathbf{r}$ is used in the construction of* $\mathsf{SpaceMac}$*. The probability that $\mathcal{A}$ wins the* $\mathsf{SpaceMac}$ *security game, denoted by* $\text{Adv}[\mathcal{A}, \mathsf{SpaceMac}]$*, 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 $\mathsf{SpaceMac}$ involves a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$, and is as follows:

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

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),
- $\mathsf{Verify}(k, \text{id}^*, \mathbf{y}^*, t^*) = 1$, 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.

Now 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,

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

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 $\mathbb{F}_q$: $r_i \overset{\text{R}}{\leftarrow} \mathbb{F}_q$ instead of $r_i \leftarrow F(k, \text{id}, i)$, 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 $(\text{id}, \mathbf{y})$. 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:
  $\mathbf{r}_i := (r^i_1, \cdots, r^i_{n+m})$, where $r^i_j \overset{\text{R}}{\leftarrow} \mathbb{F}_q, j \in [1, n+m]$
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 $(\text{id}^*, \mathbf{y}^*, t^*)$. When $\mathbf{y}^*$ does not equal $\mathbf{0}$, to determine if $\mathcal{A}$ wins the game, we compute

```
if $\text{id}^* = \text{id}_i$ (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 $r^*_i \overset{\text{R}}{\leftarrow} \mathbb{F}_q, i \in [1, n+m]$
```

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

$$t^* = \mathbf{y}^* \cdot \mathbf{r}^*\,, \text{ and} \tag{6}$$

$$\mathbf{y}^* \notin \mathsf{span}(\mathbf{y}^*_1, \cdots, \mathbf{y}^*_l)\,. \tag{7}$$

Subsequently, 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 $\mathbb{F}_q$. The given prior knowledge, the queries, and the output form the following system of linear equations Π_2:

$$(\Pi_2) \quad \begin{cases} \bar{\mathbf{p}}_1 \cdot \bar{\mathbf{r}}^* = p_1 \\ \cdots \\ \bar{\mathbf{p}}_{n-1} \cdot \bar{\mathbf{r}}^* = p_{n-1} \\ \mathbf{y}_1^* \cdot \mathbf{r}^* = t_{\mathbf{y}_1^*} \\ \cdots \\ \mathbf{y}_l^* \cdot \mathbf{r}^* = t_{\mathbf{y}_l^*} \\ \mathbf{y}^* \cdot \mathbf{r}^* = t^* \end{cases} .$$

Let the elements $r_i^*, i \in [1, n+m]$, of $\mathbf{r}^*$ be the unknowns of the system. The above system is consistent regardless of the value of t^* because the coefficient matrix has rank at most $n+m$, which equals the number of unknowns. Let d be the rank of the coefficient matrix, $d \leq n+m$. For a fixed $\mathbf{y}^*$, its valid tag t^* could be any value in $\mathbb{F}_q$ equally likely because (i) for any value t^*, the solution space always has the same size q^{n+m-d}, and (ii) 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.

Lemma 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: Lemma 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 $\mathbb{F}_q^{n+m}$.

Case (a) - The storage node is able to compute a correct response block even when some blocks are missing or corrupted: 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, $\mathsf{Prob}[\mathbf{a} \in \mathbf{\Pi}] = \mathsf{Prob}[\mathbf{a} = \mathbf{e}] \leq \frac{1}{q}$.

Case (b) - The storage node responds with an incorrect block: 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 $\frac{1}{q}$. 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 $\frac{1}{q}$.

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

Not only does NC-Audit provide detection in the presence of corrupted or missing blocks, it also ensures that the user can extract the data stored on the storage node just by collecting responses of the node from the checking protocol. This is also known as the *retrievability* property. We provide the proof of retrievability based on the theoretical framework of [19], which is derived from [15] and [16].

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

Proof: Lemma 5 implies that if a node responds 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. ■

B. Privacy-Preserving Guarantee

NCrypt provides the privacy guarantee of NC-Audit, which we stated in the following lemma.

Lemma 7. *From the responses 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 elements are chosen randomly. ■

VII. Performance Evaluation

A. 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 repairs while the TPA needs the coefficients to carry out audits. In any case, the overhead of $O(mMN)$ is orders of magnitude less than the outsourced data, which is in $O((n+m)MN)$ space; this is because $n \gg m$ for NC-based storage systems. In fact, in a practical NC storage cloud, the space necessary for storing the coding coefficients could be kept less than 160 B (*i.e.*, constant storage) while being able to support arbitrary file size (by increasing the block size n, see Section 5.1 of NCCloud [33]). Table I compares client storage overhead of NC-Audit and other state-of-the-art schemes [13], [17], [23].

B. 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 [13], [15], [17], [23]. In particular, the proof of possession for multiple blocks contains only a single block (of size varying from 4 KB [10] to 1.6 MB [13]).

Repairing. As discussed in Section V, when using NC-Audit, the user does not need to download any data block to repair a failed node. This stands in stark contrast with the state-of-the-art scheme for NC storage systems [13], where the user needs to download an amount of data equal to the repair bandwidth to setup integrity metadata for the new coded blocks him/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 [10]); the auxiliary tag, p, and the random padding elements, $e^{(n-1)}, e^{(n)}$, which are of size $\log_2 q$. These are negligible compared to the block size: $n \log_2 q$, *e.g.*, 0.3% for $q = 2^8, n = 4 \times 2^{10}$ (4 KB block).

The bandwidth overhead of NC-Audit when compared to other schemes [13], [17], [23] are summarized in Table I.

C. Computational Overhead

We first analyze the cost of each operation in NC-Audit by the number of finite field 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.

		Wang 2009 [23]	**Wang 2010 [17]**	**Chen 2010 [13]**	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
		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*
	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 I

Comparison 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.

Integrity Checking with Encryption:

1. Storage Node Overhead: In NC-Audit, the cost to compute a 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 a precomputation of $\bar{\mathbf{m}}$ and $C \times n + (n-2) \times (n-1)$ 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 (to increase the security to $1/q^{\ell}$). The total cost is $C \times m + \ell \times (n+m)$ multiplications.

Repairing:

As described in Section V, repairing a failed node does not incur any computation cost at the user side to maintain the security metadata of the auditing.

Implementation:

We implement NC-Audit in Java to compare its performance with the state-of-the-art schemes [13], [17], [23]. For a fair of comparison with [17], [23], 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 [33], a block size could be as big as 4 MB while the storage space taken by the coefficients could be kept below 160 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 I compares the computational overhead of different remote data integrity checking schemes. The reported numbers for [23] and [17] are taken from [17]. (The overhead of the scheme in [23] is similar to the public-key based scheme in [15].) We refer the reader to [17] for the detailed setup. We implement the checking scheme in [13] ourselves. For this scheme, we use AES with CBC mode from the Java *crypto* library to decrypt coding coefficients. We refer the reader to Appendix A in [13] for the detailed description of this scheme. The number reported for NC-Audit and the scheme in [13] are the average of 100 runs on a computer with 2.8 Ghz CPU and 32 GB RAM. We note that among the three schemes under comparison [13], [17], [23], the scheme in [13] is the only one specifically designed for NC storage systems and thus supports NC repair.

Table I shows that NC-Audit manages to achieve very modest computational overhead. The computational overhead of NC-Audit is orders of magnitude smaller than those of [23] and [17]. This is due to the fact that NC-Audit is symmetric-key based while the schemes in [17] and [23] are public-key based and make heavily use of expensive bilinear mapping operations[3]. The scheme in [13] achieves similar storage node's computational overhead to NC-Audit 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 of [13] is much larger

[3] Due to the fundamental difference: the use of expensive bilinear mapping operations in [17], [23], we expect a similar gap (in order of magnitude) between the computational overhead of [17], [23] and that of NC-Audit when we run them on the same hardware.

than that of NC-Audit, in the order of seconds as opposed to milliseconds.

VIII. Conclusion

In this work, we propose NC-Audit, a cryptography-based remote data integrity checking scheme for NC-based storage systems. NC-Audit is based on a novel combination of an existing MAC scheme custom made for network coding, SpaceMac, and a novel CPA-secure encryption scheme, NCrypt, which we carefully design in this work to work in synergy with SpaceMac. To the best of our knowledge, NC-Audit is the first scheme that efficiently supports auditing for NC storage systems. NC-Audit also provides protection against leakage of the outsourced data when the audit is done by a third party. Our evaluation results based on a real implementation in Java demonstrate that NC-Audit is significantly more efficient than the state-of-the-art schemes.

References

[1] D. For, F. Labelle, F.I. Popovici, M. Stokely, V.A. Truong, L. Barroso, C. Grimes, and S. Quinlan, "Availability in globally distributed storage systems," in *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, Oct. 2010.

[2] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin, "Erasure coding in windows azure storage," in *USENIX Annual Technical Conference (USENIX ATC) (Best Paper Award)*, Boston, Jun. 2012.

[3] M. Sathiamoorthy, M. Asteris, D. Papailiopoulos, A. G. Dimakis, R. Vadali, S. Chen, "XORing Elephants: Novel Erasure Codes for Big Data," (to appear) in *Very Large Data Bases (VLDB)*, Riva del Garda, Trento, Aug. 2013.

[4] Cleversafe, "Limitless Data Storage," http://www.cleversafe.com/

[5] Wuala, "Secure Cloud Storage," http://www.wuala.com/

[6] 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.

[7] 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.

[8] 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 FAST*, San Jose, CA, Feb. 2008, pp. 127–141.

[9] 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.

[10] 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.

[11] 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

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

[13] 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.

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

[15] 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.

[16] 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.

[17] 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)*, Mar. 2010, pp. 1–9.

[18] 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.

[19] 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.

[20] K. D. Bowers, A. Juels, and A. Oprea, “HAIL: A High-Availability and Integrity Layer for Cloud Storage,” in *ACM Conference on Computer and Communication Security (CCS)*, Chicago, IL, Nov. 2009, pp. 187–198.

[21] B. Wang, B. Li, and H. Li, “Public Auditing for Shared Data with Efficient User Revocation in the Cloud,” in *IEEE International Conference on Computer Communications (INFOCOM)*, Turin, Apr. 2013.

[22] 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.

[23] Q. Wang, C. Wang, J. Li, K. Ren, and W. Lou, “Enabling Public Verifiability and Data Dynamics for Storage Security in Cloud Computing,” in *ESORICS*, Saint Malo, Sep. 2009, pp. 355–370.

[24] 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)*, Mar. 2010, pp. 1–9.

[25] C. Wang, Q. Wang, K. Ren, N. Cao, and W. Lou, “Towards Secure and Dependable Storage Services in Cloud Computing,” *IEEE Transactions on Services Computing*, vol. 5, no. 2, pp. 220–232, Apr. 2012.

[26] 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.

[27] 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.

[28] ——, “On Detecting Pollution Attacks in Inter-Session Network Coding,” in *INFOCOM'12*. Mar. 2012, pp. 343–351.

[29] T.S.J. Schwarz and E.L. Miller, “Store, Forget, and Check: Using Algebraic Signatures to Check Remotely Administered Storage,” in *IEEE International Conference on Distributed Computing Systems (ICDCS)*, Lisboa, July 2006, pp. 12.

[30] A. Le, “Auditing for Distributed Storage Systems,” *Technical Report*. [Online]. Available: http://arxiv.org/abs/1203.1730

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

[32] Y. Hu, C.M. Yu, Y.K. Li, P.P.C. Lee, and J.C.S. Lui, “NCFS: On the Practicality and Extensibility of a Network-Coding-Based Distributed File System,” in *IEEE International Symposium on Network Coding (NetCod)*, July 2011, pp. 1–6.

[33] 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.

[34] N. Cao, S. Yu, Z. Yang, W. Lou, and Y.T. Hou, “LT Codes-Based Secure and Reliable Cloud Storage Service,” in *IEEE International Conference on Computer Communications (INFOCOM)*, Orlando, Mar. 2012, pp. 693–701.

[35] C. Gkantsidis and P. Rodriguez, "Cooperative security for network coding file distribution," in *IEEE International Conference on Computer Communications (INFOCOM)*, Barcelona, Apr. 2006, pp. 1–13.

[36] S. Agrawal and D. Boneh, "Homomorphic MACs: MAC-based integrity for network coding," in *ACNS'09*.

[37] 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.

[38] 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.

[39] 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 INFOCOM*, Apr. 2011, pp. 1026–1034.

[40] 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.

[41] ——, "Cooperative Defense Against Pollution Attacks in Network Coding Using SpaceMac," in *IEEE JSAC 2012*.

[42] 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.

[43] 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.

[44] 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.

[45] 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)*, Jun. 2010, pp. 66–73.

[46] A. Le and A. Markopoulou, "NC-Audit: Auditing for Network Coding Storage," in *IEEE International Symposium on Network Coding (NetCod)*, Cambridge, MA, Jun. 2012, pp. 155–160.

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

[48] 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.

[49] M. Blaum, J. Brady, J. Bruck, and J. Menon, "EVENODD: An Optimal Scheme for Tolerating Double Disk Failures in RAID Architectures," in *IEEE International Symposium on Computer Architecture (ISCA)*, Chicago, Apr. 1994, pp. 245–254.

APPENDIX

SUPPORT FOR DATA DYNAMICS

NC-Audit supports data dynamics and does not require data block download (blockless) in all operations. The approach taken by NC-Audit is similar to [18] but different from [22] and [23]: NC-Audit fully supports block append and update operations, while relying on these two operations to further support insert and delete operations. Fully supporting all operations, as in [22] and [23], come with a higher client and server computation as well as communication overhead. This is because additional data structures, such as a skip-list [22] or a binary tree

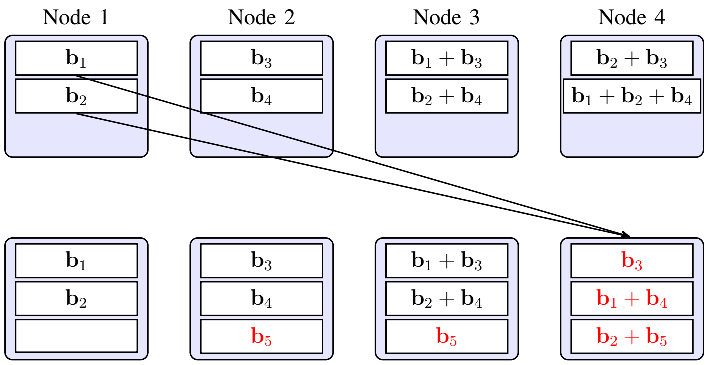
Node 1
Node 2
Node 3
Node 4
$\mathbf{b}_1$
$\mathbf{b}_2$
$\mathbf{b}_3$
$\mathbf{b}_4$
$\mathbf{b}_1 + \mathbf{b}_3$
$\mathbf{b}_2 + \mathbf{b}_4$
$\mathbf{b}_2 + \mathbf{b}_3$
$\mathbf{b}_1 + \mathbf{b}_2 + \mathbf{b}_4$
$\mathbf{b}_1$
$\mathbf{b}_2$
$\mathbf{b}_3$
$\mathbf{b}_4$
$\mathbf{b}_5$
$\mathbf{b}_1 + \mathbf{b}_3$
$\mathbf{b}_2 + \mathbf{b}_4$
$\mathbf{b}_5$
$\mathbf{b}_3$
$\mathbf{b}_1 + \mathbf{b}_4$
$\mathbf{b}_2 + \mathbf{b}_5$

Fig. 3. Appending a new block, $\mathbf{b}_5$, to an existing coded storage system (on top, as in Fig 1). The new system can still tolerate any two-node failure by leveraging an EVENODD code [49]. The user needs to upload $\mathbf{b}_5$ to nodes 2, 3, and 4, and instruct node 1 to sends to $\mathbf{b}_1$, $\mathbf{b}_2$, and their MAC tags to node 4.

[23], must be maintained. We choose the simpler approach since data modification is typically of limited use for coded storage systems, as discussed in Section II-A.

Block Append. Assume that the user wants to append a block, $\hat{\mathbf{b}}_*$, to the system. The coded blocks stored at the nodes are now a linear combination of the original source blocks, $\mathbf{b}_1, \cdots, \mathbf{b}_m$, and the new block $\mathbf{b}_*$. The encoded blocks stored at each node are updated based on the coding scheme used to attain the required level of reliability.

For instance, Fig. 3 shows how a new block, $\mathbf{b}_5$, could be added to an existing storage system (on top, as in Fig 1), where the new system can still tolerate any two-node failure by leveraging an EVENODD code [49]. Note that coded blocks at node 4 are completely changed. One way the user could achieve the new system is by instructing node 1 to send $\mathbf{b}_1$, $\mathbf{b}_2$, and their MAC tags to node 4 and also sending $\mathbf{b}_5$ to nodes 2, 3, and 4 him/herself.

We focus our discussion on how security metadata can be maintained correctly and efficiently and assume that an appropriate update scheme for the data is in place, as shown in Fig. 3. When an append is needed, the encoded $\mathbf{b}_*$ has the following form:

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

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

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

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

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

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, *i.e.*, 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 sends the new coding coefficients of the new coded blocks stored at the nodes to the TPA. 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 $\mathsf{VerifyProof}$ 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 $\mathsf{SpaceMac}$, $\mathsf{VerifyProof}$ will fail w.h.p.

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 $\mathbf{b}'_j$. To update the data, it needs to send $\mathbf{b}'_j$ to nodes that store coded blocks involving $\mathbf{b}_j$. For example, to update $\mathbf{b}_3$ in Fig. 1, the user needs to send $\mathbf{b}'_3$ to the second, third, and fourth storage nodes so that they can update $\mathbf{b}_3$, $\mathbf{b}_1 + \mathbf{b}_3$, and $\mathbf{b}_2 + \mathbf{b}_3$, respectively.

To update the security metadata, the user first needs to learn 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_v using Mac. Finally, it sends the difference between $t_{\mathbf{b}'_j}$ and $t_{\mathbf{b}_j}$: $\delta_j = t_{\mathbf{b}'_j} - t_{\mathbf{b}_j}$, to the TPA.

Subsequently, whenever challenging a storage node and obtaining a response block which involves $\alpha_j \, \mathbf{b}_j$, the TPA runs $\mathsf{VerifyProof}$ with the tag $t + \alpha_j \delta_j$ 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 $t' = \alpha_j \, t_{\mathbf{b}'_j} + \sum_{i=1,\cdots,M; i \neq j} \alpha_i \, t_{\mathbf{b}_i} = t + \alpha_j \delta_j$.

Note that if $\hat{\mathrm{e}}$ is not updated correctly by the storage node then by the security guarantee of $\mathsf{SpaceMac}$, w.h.p. t' is not a valid tag for $\mathbf{e}$. Subsequent updates to this j-th block can be carried out similarly.

This approach requires the TPA to store one field symbol δ_j for every updated source block $\mathbf{b}_j$, which is $O(m)$. This space overhead is negligible and could be constant in practice as discussed in Section VII-A. 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 corrupted 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 $O(m)$ (still negligible); 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 Insert. Similar to [18], 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.

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 $\mathsf{NC\text{-}Audit}$. Similar to [18], we consider deletion of a block as changing it to a special block. Thus, deleting a block can be done as in the *Block Update* case.