NC-Audit: Auditing for Network Coding Storage

Anh Le, Athina Markopoulou
University of California, Irvine
{*anh.le, athina*}@*uci.edu*

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

I. Introduction

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

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

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

This work was supported by NSF CAREER award 0747110 and by AFOSR MURI award FA9550-09-1-0643.

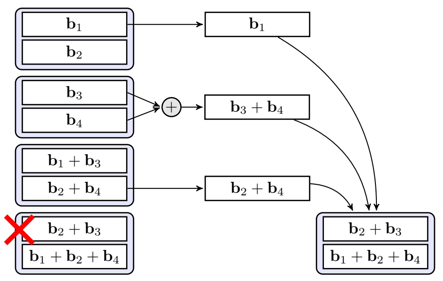
$\mathbf{b}_1$
$\mathbf{b}_2$
$\mathbf{b}_3$
$\mathbf{b}_4$
$\mathbf{b}_1 + \mathbf{b}_3$
$\mathbf{b}_2 + \mathbf{b}_4$
$\mathbf{b}_2 + \mathbf{b}_3$
$\mathbf{b}_1 + \mathbf{b}_2 + \mathbf{b}_4$
$\mathbf{b}_1$
$\mathbf{b}_3 + \mathbf{b}_4$
$\mathbf{b}_2 + \mathbf{b}_4$
$\mathbf{b}_2 + \mathbf{b}_3$
$\mathbf{b}_1 + \mathbf{b}_2 + \mathbf{b}_4$

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

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

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

(i) **Efficient Integrity Checking**: The integrity check incurs a small bandwidth and computation overhead (few milliseconds). It guarantees that, with high probability, the storage provider passes the integrity check if and only if it possesses the data. The proposed protocol also supports unlimited number of checks.
(ii) **Efficient Support for Repair and Data Dynamics**: The repair of failed nodes and the changes made to the data (including update, append, insert, and delete operations) require negligible bandwidth (no data download) and computation (sub milliseconds) to maintain the metadata used by the integrity checking.
(iii) **Efficient Privacy Protection**: A third party auditor cannot learn any information about the user data through the checking protocol, except for the metadata used by the integrity checking. This privacy preserving property incurs a small bandwidth (0.4%) and computation overhead (few milliseconds).

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

We implemented NC-Audit in Java, utilizing our previous

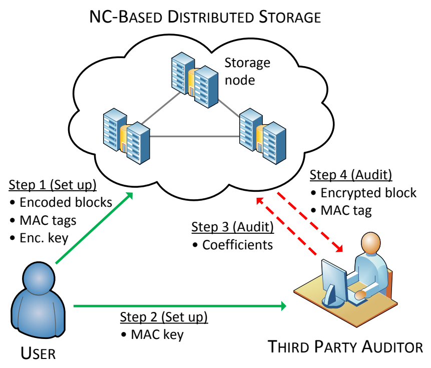

Fig. 2. Parties and Steps Involved in NC-Audit.

implementation of SpaceMac [20]. Our evaluation of NC-Audit shows that it has very low computation overhead: when performing an audit, both the storage node and the TPA only need to spend a couple of milliseconds.

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 (SpaceMac and NCrypt) before presenting NC-Audit itself. We also show how NC-Audit efficiently supports repair and data dynamics. In Section V, we analyze the security of NC-Audit. In Section VI, we evaluate its bandwidth and computational efficiency. In Section VII, we conclude.

II. Related Work

The benefits of network coding for distributed storage has been first formalized by the work of Dimakis *et al.* [2]. An excellent survey on recent advances in NC-based storage systems can be found at [1]. One of the first implementations of NC-based storage cloud is NCCloud by Hu *et al.* [21]. A wiki on NC-based storage cloud is maintained at [22].

In [19], Dikialotis *et al.* proposed an integrity checking scheme for NC-based storage cloud which requires a very small amount of bandwidth. The key technique for reducing the amount of bandwidth is to project blocks on a small random vector. This technique requires communicating with multiple nodes to perform a single check while our work does not.

In [18], Chen *et al.* proposed a cryptographic integrity checking scheme for NC-based storage. This scheme adopts the symmetric-key based integrity checking scheme that Shacham and Waters [9] proposed for regular cloud storage with minor modification, thus not exploiting network coding for efficient checking. In addition, the scheme in [18] neither supports data dynamics nor privacy-preserving auditing.

There has been a rich body of work on integrity checking for remote data [5]–[7], [9]–[17], known as *Proof of Retrievability* or *Proof of Data Possession*. These works, however, are not customized for NC-based storage systems and do not efficiently support coding operations and repair of failed nodes. Other security problems for NC-based storage include securing blocks when repairing [23]–[26] as well as defense against pollution attacks [27].

III. Problem Formulation

A. System Model and Operations

Fig. 2 illustrates an overview of the system. We consider a cloud storage service that involves three entities: a user, NC-based storage nodes, which make up the storage cloud, and a third party auditor (TPA). The user distributes her data on the storage nodes and may also dynamically update her data. The user resorts to a TPA to check for the integrity of her data stored at each node; at the same time, she does not want the TPA to learn about her data. We adopt the storage model in [21] where the user is responsible for maintaining the data stored at each storage node. Our work, however, is also applicable to the scenario where there is a cloud service provider who is independent from the user and is responsible for maintaining the storage cloud.

The user follows the following basic steps to store her data on the storage cloud. We adopt the notations used in [28]. 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 augments each block $\hat{\mathbf{b}}_i$ with its m *global coding coefficients*. The resulting blocks, $\mathbf{b}_i$, have the following form:

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

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 the n-extended version of a storage code [19].

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

B. Threat Model

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

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

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

IV. Auditing Scheme

A. Definitions and Auditing Framework

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

- $\mathsf{KeyGen}(1^\lambda) \rightarrow (k_1, k_2)$ is a probabilistic key generation algorithm that is run by the user to setup the scheme. It takes a security parameter, λ, as input and outputs two different private keys, k_1 and k_2. k_1 is used to generate verification metadata, and k_2 is used to encrypt the possession proof.
- $\mathsf{TagGen}(\mathbf{e}, k_1) \rightarrow t$ is a probabilistic algorithm run by the user to generate the verification metadata. It takes as input a coded block, $\mathbf{e}$, a private key, k_1, and outputs a verification data of $\mathbf{e}$, t.
- $\mathsf{GenProof}(k_2, (\mathbf{e}_1, \cdots, \mathbf{e}_M), (t_{\mathbf{e}_1}, \cdots, t_{\mathbf{e}_M}), \mathsf{chal}) \rightarrow V$ is run by the storage node to generate a proof of possession. It takes as input a secret key, k_2, coded blocks stored at the node, $\mathbf{e}_1, \cdots, \mathbf{e}_M$, their corresponding verification metadata, $t_{\mathbf{e}_1}, \cdots, t_{\mathbf{e}_M}$, and a challenge, chal. It outputs a proof of possession, V, for the coded blocks specified in chal.
- $\mathsf{VerifyProof}(k_1, \mathsf{chal}, V) \rightarrow \{1, 0\}$ is run by the user in order to validate a proof of possession. It takes as inputs a secret key k_1, a challenge, chal, and a proof of possession V. It returns 1 (success) if V is the correct proof of possession for the blocks specified in chal and 0 (failure) otherwise.

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

- *Setup*: The user initializes the security parameters of the system by running KeyGen. The encoded blocks are prepared as described in Section 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 GenProof and sends the possession proof back to the TPA. Finally, the TPA runs VerifyProof to verify the possession proof it receives.

B. Basic Scheme and Key Techniques

Next, we describe the basic scheme [5] and then describe how we improve this basic scheme to arrive at our proposed scheme.

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

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

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

In particular, we adopt SpaceMac, a homomorphic MAC scheme that we previously designed specifically for network coding [20], [29]. 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 a response block, even when encrypted, can be used by the TPA for the integrity check. NCrypt employs the random linear combination technique of network coding to be compatible with SpaceMac verification. NCrypt is semantically secure under a chosen plaintext attack (CPA-secure). Next, we describe SpaceMac and NCrypt in detail.

C. The Homomorphic MAC: SpaceMac

In prior work, we designed SpaceMac and used it to combat pollution attacks in network coding [20], [28]–[30]. Here, we use SpaceMac to support the aggregation of file blocks and tags. SpaceMac consists of a triplet of algorithms: Mac, Combine, and Verify. The construction of SpaceMac uses a pseudo-random function (PRF) $F_1 : \mathcal{K}_1 \times (\mathcal{I} \times [1, n+m]) \rightarrow \mathbb{F}_q$, where $\mathcal{K}_1$ is the PRF key domain and $\mathcal{I}$ is the file identifier domain.

- $\mathsf{Mac}(k, \mathsf{id}, \mathbf{e}) \rightarrow t$: The MAC tag $t \in \mathbb{F}_q$ of a source block or encoded block, denoted by $\mathbf{e} \in \mathbb{F}_q^{n+m}$, under key k, can be computed by the following steps:
  – $\mathbf{r} \leftarrow (F_1(k, \mathsf{id}, 1), \cdots, F_1(k, \mathsf{id}, n+m))$.
  – $t \leftarrow \mathbf{e} \cdot \mathbf{r} \in \mathbb{F}_q$.
- $\mathsf{Combine}((\mathbf{e}_1, t_1, \alpha_1), \cdots, (\mathbf{e}_\ell, t_\ell, \alpha_\ell)) \rightarrow t$: The tag $t \in \mathbb{F}_q$ of $\mathbf{e} \overset{\text{def}}{=} \sum_{i=1}^{\ell} \alpha_i \mathbf{e}_i \in \mathbb{F}_q^{n+m}$ is computed as follows:
  – $t \leftarrow \sum_{i=1}^{\ell} \alpha_i t_i \in \mathbb{F}_q$.
- $\mathsf{Verify}(k, \mathsf{id}, \mathbf{e}, t) \rightarrow \{0, 1\}$: To verify if t is a valid tag of $\mathbf{e}$ under key k, we do the following:
  – $\mathbf{r} \leftarrow (F_1(k, \mathsf{id}, 1), \cdots, F_1(k, \mathsf{id}, n+m))$.
  – $t' \leftarrow \mathbf{e} \cdot \mathbf{r}$.
  – If $t' = t$, output 1 (accept); otherwise, output 0 (reject).

Lemma 1 (Theorem 1 in [29]). *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 [29] for the security game and proof of SpaceMac. If the user computes the verification tags for the source blocks using Mac, then the storage node can compute a valid MAC tag for any encoded block using Combine. The security of SpaceMac guarantees that if a block, $\mathbf{e}'$, is not a linear combination of the source blocks, then the storage node can only forge a valid MAC tag for $\mathbf{e}'$ with probability $\frac{1}{q}$. The security when using ℓ tags is $\frac{1}{q^\ell}$. Also, 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. Here, we describe NCrypt, an encryption scheme that is compatible with SpaceMac. In particular, NCrypt will protect $n-1$ elements of the response block while still allowing SpaceMac integrity checking. Only $n-1$ elements rather than n is protected is because of the technical constraint needed to preserve the security guarantee of SpaceMac.

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

- $\mathsf{Setup}(k, \bar{\mathbf{r}}) \to (p_1, \cdots, p_{n-1})$ run by the user to setup the encryption scheme. It takes as input a secret key, k, and a vector, $\bar{\mathbf{r}} \in \mathbb{F}_q^{n-1}$. It outputs $n-1$ elements in $\mathbb{F}_q$, which are called *tagging elements* and are used by the encryption. The details are as follow:
  – $\bar{\mathbf{p}}_i \leftarrow (F_2(k, i, 1), \cdots, F_2(k, i, n-1))$, for $i \in [1, n-1]$.
  – $p_i \leftarrow \bar{\mathbf{r}} \cdot \bar{\mathbf{p}}_i$, for $i \in [1, n-1]$.
- $\mathsf{Enc}(k, \bar{\mathbf{e}}, (p_1, \cdots, p_{n-1})) \to \langle \bar{\mathbf{c}}, (r, p) \rangle$ run by the storage node to encrypt $n-1$ first elements of the response block. It takes as input a secret key, k, vector formed by the first $n-1$ elements of the response block, $\bar{\mathbf{e}}$, and the tagging elements, $p_1, \cdots, p_{n-1}$. It computes the encryption, $\langle \bar{\mathbf{c}}, (r, p) \rangle$, of $\bar{\mathbf{e}}$ as follows:
  – Compute $\bar{\mathbf{p}}_i, i \in [1, n-1]$, using key k as in Setup.
  – Choose r uniformly at random: $r \overset{R}{\leftarrow} \{0,1\}^\lambda$.
  – Compute the masking coefficients:
  $\beta_i \leftarrow F_3(k, r, i) \in \mathbb{F}_q$, for $i \in [1, n-1]$.
  – Compute masking vector: $\bar{\mathbf{m}} \leftarrow \sum_{i=1}^{n-1} \beta_i \bar{\mathbf{p}}_i \in \mathbb{F}_q^{n-1}$.
  – Compute $\bar{\mathbf{c}} \leftarrow \bar{\mathbf{e}} + \bar{\mathbf{m}} \in \mathbb{F}_q^{n-1}$.
  – Compute $p \leftarrow \sum_{i=1}^{n-1} \beta_i p_i$.
  In essence, the data is masked with a randomly chosen vector $\bar{\mathbf{m}} \in \mathsf{span}(\bar{\mathbf{p}}_1, \cdots, \bar{\mathbf{p}}_{n-1})$.
- $\mathsf{Dec}(k, \langle \bar{\mathbf{c}}, (r, p) \rangle) \to \bar{\mathbf{e}}$ takes as input a secret key, k, and the cipher text, $\langle \bar{\mathbf{c}}, (r, p) \rangle$. The decryption is done as follows:
  – Compute $\bar{\mathbf{p}}_i, i \in [1, n-1]$, using key k as in Setup.
  – Compute $\beta_i \leftarrow F_3(k, r, i) \in \mathbb{F}_q$, for $i \in [1, n-1]$.
  – Compute $\bar{\mathbf{m}} \leftarrow \sum_{i=1}^{d} \beta_i \bar{\mathbf{p}}_i \in \mathbb{F}_q^{n-1}$.
  – Compute $\bar{\mathbf{e}} \leftarrow \bar{\mathbf{c}} - \bar{\mathbf{m}} \in \mathbb{F}_q^{n-1}$.

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

Intuitively, the security of NCrypt holds because $\bar{\mathbf{m}}$ looks completely random to an adversary who observes a ciphertext $\langle \bar{\mathbf{c}}, (r, p) \rangle$ since it is computationally difficult for the adversary to compute $\bar{\mathbf{p}}_i$'s and β_i's without knowing the secret key k. We refer the reader to [31] for the details of the proof.

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

Our symmetric-key based auditing protocol, denoted by NC-Audit, is built from SpaceMac and NCrypt as follows:

- *Setup* phase:
  – The user divides the file into m blocks of size $n-1$ instead of n and pads to each block a random element in $\mathbb{F}_q$. This is necessary as NCrypt encrypts only the first $n-1$ elements. We still denote each padded block with its coding coefficients by $\mathbf{b}_i, i \in [1, m]$.
  – The user runs KeyGen to generate MAC key, k_1, and encryption key, k_2:
  ∘ $\mathsf{KeyGen}(1^\lambda) \to (k_1, k_2)$: $k_1, k_2 \overset{R}{\leftarrow} \{0,1\}^\lambda$.
  – The user then setups the encryption scheme by computing the tagging elements, $p_1, \cdots, p_{n-1}$:
  ∘ $\bar{\mathbf{r}} \leftarrow (F_1(k_1, 1), \cdots, F_1(k_1, n-1))$.
  ∘ $(p_1, \cdots, p_{n-1}) \leftarrow \mathsf{Setup}(k_2, \bar{\mathbf{r}})$.
  – Afterward, the user computes a tag for each source block $\mathbf{b}_i$ using Mac algorithm of SpaceMac:
  ∘ $t_{\mathbf{b}_i} = \mathsf{Mac}(k_1, \mathbf{b}_i)$.
  – MAC tags of encoded blocks are computed by the Combine algorithm of SpaceMac: Assume $\mathbf{e} = \sum_{i=1}^{m} \alpha_i \mathbf{b}_i$, then
  ∘ $\mathsf{TagGen}(\mathbf{e}, k_1) \to t_\mathbf{e}$: $t_\mathbf{e} = \sum_{i=1}^{m} \alpha_i t_{\mathbf{b}_i}$.
  – Finally, the user sends the encoded blocks, $\mathbf{e}_1, \cdots, \mathbf{e}_M$, their tags, $t_{\mathbf{e}_1}, \cdots, t_{\mathbf{e}_M}$, the tagging elements, $p_1, \cdots, p_{n-1}$, and the encryption key, k_2, to the storage node. The user also sends the coding coefficients, $\mathsf{aug}(\mathbf{e_1}), \cdots, \mathsf{aug}(\mathbf{e_M})$, and the MAC key, k_1, to the TPA. We assume that the user uses private and authentic channels to send k_1 and k_2 while using an authentic channel for sending the other data. The user then keeps the keys and the coding coefficients (for repair) but delete all other data. Note that the storage overhead of the user and the TPA is $O(mMN)$, which is negligible compared to the outsourced data $O((n+m)MN)$ since $n \gg m$.
- *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:
  ∘ $\mathsf{chal} = \{(i, \alpha_i) | i \in \mathcal{I}\}$.
  – GenProof run by the node to generate the proof of storage, V, is implemented as follows:
  ∘ Compute the aggregated block: $\hat{\mathbf{e}} = \sum_{i \in \mathcal{I}} \alpha_i \hat{\mathbf{e}}_i$. Parse $\hat{\mathbf{e}}$ as $(\bar{\mathbf{e}}, e^{(n)})$.
  ∘ Compute the aggregated tag: $t = \sum_{i \in \mathcal{I}} \alpha_i t_{\mathbf{e}_i}$.
  ∘ Encrypt block: $\langle \bar{\mathbf{c}}, (r, p) \rangle \leftarrow \mathsf{Enc}(k_2, \bar{\mathbf{e}}, (p_1, \cdots, p_{n-1}))$.
  The node then sends $V = (\langle \bar{\mathbf{c}}, (r, p) \rangle, e^{(n)}, t)$ back to the TPA.
  – VerifyProof run by the TPA to verify the proof V is implemented as follows:
  ∘ Compute coefficients of $\hat{\mathbf{e}}$: $\mathsf{aug}(\mathbf{e}) = \sum_{i \in \mathcal{I}} \alpha_i \, \mathsf{aug}(\mathbf{e_i})$.
  ∘ Let $\mathbf{c} = (\bar{\mathbf{c}} \,|\, e^{(n)} \,|\, \mathsf{aug}(\mathbf{e}))$, "|" denotes augmentation.
  ∘ Return result of $\mathsf{Verify}(k_1, \mathbf{c}, t+p)$.

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

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

Proof: Let $\mathbf{r} = (F_1(k, 1), \cdots, F_1(k, n+m))$. Note that $\mathbf{c} = (\bar{\mathbf{c}} \,|\, e^{(n)} \,|\, \mathsf{aug}(\mathbf{e})) = ((\bar{\mathbf{e}} + \bar{\mathbf{m}}) \,|\, e^{(n)} \,|\, \mathsf{aug}(\mathbf{e})) = \mathbf{e} + (\bar{\mathbf{m}} \,|\, 0, \cdots, 0)$. Thus, in the Verify, $t' = \mathbf{c} \cdot \mathbf{r} = \mathbf{e} \cdot \mathbf{r} + \bar{\mathbf{m}} \cdot \bar{\mathbf{r}} = t + \sum_{i=1}^{n-1} \beta_i \bar{\mathbf{p}}_i \cdot \bar{\mathbf{r}} = t + \sum_{i=1}^{n-1} \beta_i p_i = t + p$. Therefore, Verify returns 1. Hence, the TPA accepts the proof. ∎

F. Efficient Support for Repair and Data Dynamics

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 nodes, the user instructs these nodes to send appropriate coded blocks to the new node. The new node then linearly combines them, according to the user instruction, to construct its own coded blocks. This new node may construct the same coded blocks that the failed node had (*exact repair*), or completely different coded blocks (*functional repair*) [1].

Using NC-Audit, the verification tags of the newly constructed blocks at the new node do not need to be computed by the user. In particular, the healthy nodes can send along the verification tags of the coded blocks that they send to the new node. The new node can use Combine to generate tags corresponding to the coded blocks that it needs to construct. As a result, with NC-Audit, there is no cost, in term of both bandwidth and computation of verification metadata, to the user when repairing a failed node.

Data Dynamics. NC-Audit efficiently supports changes that the user may want to make to their outsourced data, including block update, block delete, block append, and block insert. The users can carry on these updates without the need of downloading data blocks. For the interest of space, we only discuss block append, which is the most important operation of NC-based storage cloud, and refer the reader to [31] for the details on how NC-Audit supports the other operations.

- *Block Append:* Assume that the user want to append a source block $\mathbf{b}_{m+1}$ to the system that has m source blocks. It first compute the tag $t_{\mathbf{b}_{m+1}}$ of $\mathbf{b}_{m+1}$ under k_1 using Mac. Then, it sends $t_{\mathbf{b}_{m+1}}$ to all storage nodes that have coded packets that involve $\mathbf{b}_{m+1}$.

  Assume a storage node has a coded packet $\mathbf{e} = \sum_{i=1}^{m} \alpha_i \mathbf{b}_i$, where $\mathbf{e} = (e_1, \cdots, e_{n+m})$, then $\mathbf{e}$ old tag is $t_{\mathbf{e}} = \sum_{i=1}^{m} e_{n+i} t_{\mathbf{b}_i}$. The representation of $\mathbf{e}$ after a block is appended to the system is $\mathbf{e}' = (e_1, \cdots, e_{n+m}, 0)$. Thus $t_{\mathbf{e}'} = \sum_{i=1}^{m} e_{n+i} t_{\mathbf{b}_i} + 0 \cdot t_{\mathbf{b}_{m+1}} = t_{\mathbf{e}}$. Assume α_{m+1} of $\mathbf{b}_{m+1}$ is added to $\mathbf{e}'$ after the append, then the storage node can compute new tag of $\mathbf{e}'' = (\mathbf{e}' \,|\, \alpha_{m+1})$: $t_{\mathbf{e}''} = t_{\mathbf{e}'} + \alpha_{m+1} t_{\mathbf{b}_{m+1}}$.

  Finally, the user must send the new coding coefficients, $\mathsf{aug}(\mathbf{e_1}), \cdots, \mathsf{aug}(\mathbf{e_M})$, of the new coded packets at the storage node to the TPA. The exact coefficients depend on how the coding scheme of the system handles append.

V. Security Analysis

A. Data Possession Guarantee

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

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

The proof of this theorem is provided in [31]. The following theorem summarizes data possession guarantee of NC-Audit.

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

This is a consequence of Theorem 4. For the interest of space, we refer the reader to [31] for the formal proof. NC-Audit actually provides a stronger data possession guarantee. It ensures that the user can extract the data stored on the storage node just by collecting responses of the node from the checking protocol. The following theorem, which states the proof of retrievability of NC-Audit, is based on the theoretical framework of [13] (derived from [10] and [9]).

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

The proof is also provided in [31].

B. Privacy-Preserving Guarantee

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

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

The claim is a direct consequence of Theorem 2 and the fact that the padding element is chosen randomly. We stress that the information derived from the MAC tags are not sufficient to derive the outsourced data. To be concrete, each tag is a weighted sum of symbols belonging to the same block. Also, the outsourced data consists of $m \times n$ field symbols, which could be considered as unknowns of a system of linear equations, and the knowledge given by the tags and the MAC key only gives n linearly independent equations.

VI. Performance Evaluation

A. 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 [7], [9], [11], [18], *i.e.*, the proof of possession for multiple blocks contains only a single block (of size varying from 4 KB [5] to 1.6 MB [18]).

Repairing and Updating: As shown in Section IV-F, when using NC-Audit, the user does not need to download any data block to repair failed nodes or update the outsourced data. This stands in stark contrast with the state-of-the-art scheme for NC storage [18]. In this scheme, the user needs to download the amount of data equal to the amount that the remaining healthy nodes need to send to the newly constructed node, *i.e.*, equal to the repair bandwidth. Furthermore, the scheme in [18] does not support data dynamics.

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

		Wang 2009 [15]	**Wang 2010 [11]**	**Chen 2010 [18]**	NC-Audit
Features		Public-key audit	Public-key audit	Symmetric-key audit	Symmetric-key audit
		No NC-based repair	*No* NC-based repair	NC-based repair	*Efficient* NC-based repair
		No data dynamics	Data dynamics	*No* data dynamics	Data dynamics
		No privacy protection	Privacy protection	*No* privacy protection	Privacy protection
Bandwidth	Audit Overhead	1 data block	1 data block	1 data block	1 data block
	Repair Overhead	N/A	N/A	Repair bandwidth for a node	0*
	Updating Overhead	0*	N/A	N/A	0*
	Encryption Overhead	N/A	0*	N/A	0*
Computation	Security	80-bit			
	Parameters	300 blocks per challenge, 4 KB block size			
	Testbed Configuration	1.86 Ghz CPU, 2GB RAM		2.8 Ghz CPU, 32 GB RAM	
	Storage Node Overhead	270 ms	273 ms	3.19 ms	4.69 ms
	TPA Overhead	491 ms	493 ms	2.76 s	0.73 ms

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

B. Computational Overhead

To evaluate the computational overhead of NC-Audit, we first analyze the dominating cost of each operation in NC-Audit. Due to lack of space, we refer the reader to [31] for the detailed analysis. Here, we present our evaluation from real implementation.

Implementation: We implement NC-Audit in Java to compare its performance with recent schemes [11], [15], [18]. For a fair comparison with [11], [15], we use $q = 2^8$ and $\ell = 10$ to provide 80-bit security and 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 implement finite field multiplications in $\mathbb{F}_{2^8}$ by using table look-ups and additions by using XORs. We use our previous Java implementation of SpaceMac [20] to compute, combine, and verify tags. We also precompute values that do not depend on the challenges, such as, PRF calls and masking vectors.

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

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

VII. Conclusion

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

References

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

[2] A. Dimakis, B. Godfrey, Y. Wu, M. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," *IEEE Tran. Info. Theory*, vol. 56, no. 9, 2010.

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

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

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

[6] M. A. Shah, R. Swaminathan, and M. Baker, "Privacy-Preserving Audit and Extraction of Digital Contents," in *Cryptology ePrint Archive, Report 2008/186*, 2008.

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

[8] Cloud Security Alliance, "Security Guidance for Critical Areas of Focus in Cloud Computing," 2012. [Online]. Available: http://goo.gl/DCOQM

[9] H. Shacham and B. Waters, "Compact Proofs of Retrievability," in *Asiacrypt*, Melbourne, Dec. 2008, pp. 90–107.

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

[11] C. Wang, Q. Wang, K. Ren, and W. Lou, "Privacy-Preserving Public Auditing for Data Storage Security in Cloud Computing," in *IEEE INFOCOM*, San Diego, 2010.

[12] G. Ateniese, R. Di Pietro, L. V. Mancini, and G. Tsudik, "Scalable and efficient provable data possession," in *SecureComm*, Istanbul, Sep. 2008, pp. 1–10.

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

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

[15] Q. Wang, C. Wang, J. Li, K. Ren, and W. Lou, "Enabling Public Verifiability and Data Dynamics for Storage Security in Cloud Computing," in *ESORICS*, 2009.

[16] S. Yu, C. Wang, K. Ren, and W. Lou, "Achieving Secure, Scalable, and Fine-grained Data Access Control in Cloud Computing," in *IEEE INFOCOM*, 2010.

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

[18] B. Chen, R. Curtmola, G. Ateniese, and R. Burns, "Remote Data Checking for Network Coding-based Distributed Storage Systems," in *ACM CCSW*, Oct. 2010, pp. 31–42.

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

[20] A. Le and A. Markopoulou, "Cooperative Defense Against Pollution Attacks in Network Coding Using SpaceMac," *(to appear) IEEE JSAC*, 2011.

[21] 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 FAST*, San Jose, CA, Feb. 2012, pp. 265–272.

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

[23] S. Pawar, S. E. Rouayheb, and K. Ramchandran, "On Secure Distributed Data Storage Under Repair Dynamics," in *IEEE ISIT*, Austin, TX, Jun. 2010, pp. 2543–2547.

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

[25] S. Pawar, S. E. Rouayheb, and K. Ramchandran, "Securing Dynamic Distributed Storage Systems from Malicious Nodes," in *IEEE ISIT*, Saint Petersburg, Jul. 2011, pp. 1452–1456.

[26] ——, "Securing Dynamic Distributed Storage Systems against Eavesdropping and Adversarial Attacks," *Tran. Info. Theory*, vol. 57, no. 9, pp. 1–19, Sep. 2011.

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

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

[29] ——, "Locating Byzantine Attackers in Intra-Session Network Coding using SpaceMac," in *IEEE NetCod*, Toronto, Jun. 2010, pp. 1–6.

[30] ——, "On Detecting Pollution Attacks in Inter-Session Network Coding," in *IEEE INFOCOM*, Orlando, FL, Mar. 2012, pp. 343–351.

[31] ——, "NC-Audit: Auditing for Network Coding Storage," *Technical Report.* [Online]. Available: http://arxiv.org/abs/1203.1730

[32] J. Katz and Y. Lindell, *Introduction to Modern Cryptography.* Chapman Halls, 2007.