Auditing for Distributed Storage Systems

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

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, 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, this comes with many drawbacks, such as limiting...
the usage of the outsourced data. We discuss this in detail in Section IV-B.

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) and negligible 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 on a computer with Intel Xeon 2.8 Ghz. 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 on a computer with Intel Core 2 1.86 Ghz.

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).** Juels and Kaliski [16] 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.

Shacham and Waters [15] 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 the existing approach [15] in that NC-Audit exploits the inherent embedded coding coefficients in the stored blocks to perform the extraction. Meanwhile, the previous work [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 by Ateniese et al. [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, Schwarz and Miller [29] proposed using a combination of both erasure-correcting coding and algebraic signatures (homomorphic hashes) to perform integrity checking for remote data. As discussed a previous work [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 [21]. We will show that NC-Audit provides the stronger data possession checking with data extraction as in POR (Section VI-A).

Data Modification. Ateniese et al. [18] 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. Erway et al. [22] proposed an auditing scheme built on rank-based authenticated skip lists and requires the storage server to maintain the lists for verification. Wang et al. [23] proposed a public auditing scheme that uses a combination of the BLS-based scheme [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 our technical report [30].

Privacy Preserving Shah et al. [11] 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. Wang et al. [17] proposed a privacy preserving auditing protocol that has public verifiability. This protocol can be considered an extension of the BLS-based protocol [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 the previous work [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, 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 is available [6]. A wiki on NC-based storage cloud is also available [31]. NC-Audit is designed to fully support regenerating codes.

An NC-based distributed file system (NCFS) is proposed by Hu et al. [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. Dikialotis et al. [26] 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 by Chen et al. [13]. In this work, the authors 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 [15], this scheme [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 the previous work [13] is that NC-Audit fully exploits network coding for integrity checking. In particular, the previous scheme [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
NC-BASED DISTRIBUTED STORAGE

![Diagram of NC-Audit process]

**Step 1 (Set up)**
- Encoded blocks
- MAC tags
- Enc. key

**Step 2 (Set up)**
- MAC key

**Step 3 (Audit)**
- Coefficients

**Step 4 (Audit)**
- Encrypted block
- MAC tag

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

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 previous scheme [13] does not support privacy-preserving auditing while NC-Audit does. We provide detailed performance comparison between NC-Audit and this scheme [13] in Section VII.

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. There has been work [42], [43] to prevent eavesdroppers from accessing/decoding all the data. In particular, Pawar et al. [42] provide an explicit code construction that achieves the secrecy capacity for the bandwidth-limited regime of the storage systems under repair dynamics. Rouayheb et al. [43] analyze the effects of interaction between the storage nodes on the amount of data revealed to the eavesdroppers. Pawar et al. [44] provide upper bounds on the maximum amount of information that can be stored safely when there are malicious nodes.

There are also work that provides protection against pollution attacks during the repair [44], [45]. In particular, Buttyan et al. [45] provide a lightweight, pollution-resilient decoding algorithm that is capable of finding adversarial blocks. Chen et al. [13] also protect 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].

**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 there is a single entity that is responsible for repairing failed nodes. This entity could be either the user him/herself or a trusted proxy that manages the storage nodes, as in the case of NCCloud [33]. In the scenario where all the nodes are provided by the same storage provider, e.g., Amazon, this storage provider could assume the repair responsibility itself. In this case, we assume that the repair scheme is known by the TPA so that it can correctly audit the newly constructed node. We discuss this in more detail in Section V. For ease of composition, in all other parts of this work, we assume the user is responsible for the repairing.

The user follows the following basic steps to store his/her data on the storage cloud. We adopt the notation used previously [40]. Denote the original file by $F$. The user first divides $F$ into $m$ blocks, $b_1, \ldots, b_m$. 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 $b_i$ with its $m$ global coding coefficients. The resulting blocks, $b_i$, have the following form:

$$b_i = \left(\begin{array}{c}
-b_{i,0}, 0, \ldots, 0, 1, 0, \ldots, 0
\end{array}\right) \in \mathbb{F}_q^{n+m}.$$

We call $b_i$ *source blocks* and the space spanned by them *source space*, denoted by $\Pi$. We use $\text{aug}(b_i)$ to denote the coefficients of $b_i$. 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 $e$ equals $\sum_{i=1}^{m} \alpha_i b_i$, then the last $m$ coordinates of $e$ are exactly the coding coefficients $\alpha_i$'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 a previously established threat model [17], [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 [17]. In order words, the auditing protocol should not introduce a data leakage vulnerability.

Finally, 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:

- **KeyGen**($\lambda$) → ($k_c$, $e_c$) is a key generation algorithm that is run by the user to set up the scheme. It takes a security parameter $\lambda$ as input and outputs two different private keys, $k_c$ and $e_c$, both in $\{0, 1\}^\lambda$:
  - $k_c$ is used to generate verification metadata and is also sent to the TPA to be used to verify the possession proofs.
  - $e_c$ is sent to the storage node to encrypt the proofs.

- **TagGen**($k_v$, $e$) → $t$ is an algorithm run by the user to generate the verification metadata. It takes as input a private key, $k_v$, a source block or encoded block, $e$, and outputs a verification tag of $e$.

- **GenProof**($k_e$, ($e_1$, ···, $e_M$), ($t_{e_1}$, ···, $t_{e_M}$), $chal$) → $V$ 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, $e_1$, ···, $e_M$; their corresponding verification metadata, $t_{e_1}$, ···, $t_{e_M}$; and a challenge, $chal$, which includes block indices and coding coefficients. It outputs a proof of possession, $V$; for the coded blocks determined by $chal$.

- **VerifyProof**($k_v$, $chal$, $V$) → {1, 0} is run by the TPA 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 GenProof, and it 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

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 was also described before [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 pre-computes a Message Authentication Code (MAC) tag, $t_i$, for each coded block, $e_i$, 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 $e_i$ correctly, the TPA issues a request for $e_i$. The node then sends $e_i$ and its tag $t_i$ to the TPA. The TPA can use $k_v$ and $t_i$ to check for the integrity of $e_i$. 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, which essentially re-sets 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.

**Encrypt All Data.** Another straightforward way to provide a privacy-preserving auditing scheme is to encrypt all data before outsourcing it, and use one of the existing integrity checking schemes [15], [16], [19] or the SpaceMac scheme, which will be described later in this work.

The main drawback of this approach is that it will severely limit the usage of the outsourced data, thus significantly reducing the application of the distributed storage system. In particular, if the data is encrypted using a standard AES encryption, the user will not be able to perform computation on her own outsourced data without first decrypting it. The computation includes searching, ranking, editing, etc. This also prevents the user from using additional services that the storage provider provides on the outsourced data, such as backup or archiving.
as searching (Amazon’s CloudSearch) and photo management (Dropbox’s Carousel).

To accommodate computation on encrypted data, advanced homomorphic encryption schemes can be used, and this is a very active research topic. However, to this date, none of the proposed homomorphic encryption schemes is suitable for everyday use yet. This is due to their large overhead of computation and/or memory usage [49], [50].

Therefore, it is desirable for an auditing scheme to support unencrypted data. In particular, it should not introduce new vulnerabilities of unauthorized data leakage when a third party performs the auditing. This desired property is also studied in many existing literature [17], [25], [51], [52], and it serves as one of our motivations to introduce NC-Audit. In this work, we consider auditing orthogonal to data encryption, and that data encryption is complementary to our proposed auditing scheme: it could be used for highly sensitive data without interfering with the operations of NC-Audit.

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

The main novelty of NCrypt lies in its compatibility with SpaceMac: It is carefully designed to maintain 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 were discussed in our previous work [27], [41]. Here, we adopt SpaceMac to support the aggregation of file blocks and tags to allow for efficient auditing (similar to existing approaches [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.

- **Mac($k$, id, e):**
  - Input: A secret key, $k$, the identifier, id, of the file, and a source block or encoded block, $e \in \mathbb{F}_q^{n+m}$.
  - Output: Tag $t$ for $e$.

  The identifier id is the unique identifier of each file. If two blocks are the same but belong to two different files, i.e., having different id’s, the Mac algorithm must generate different tags even when using the same key. This is to prevent the replay attack, where an adversary may break the security of the MAC scheme by reusing known tags. More detail about the role of the identifier can be found in Section VI-A.

- **Combine(($e_1$, $t_1$, $\alpha_1$), \ldots, ($e_t$, $t_e$, $\alpha_e$)):**
  - Input: $\ell$ blocks, $e_1, \ldots, e_\ell$, their tags, $t_1, \ldots, t_\ell$, under key $k$, and their coefficients, $\alpha_1, \ldots, \alpha_\ell \in \mathbb{F}_q$.
  - Output: Tag $t$ for block $e \overset{\text{def}}{=} \sum_{i=1}^\ell \alpha_i e_i$.

  The Combine algorithm is the key property of a homomorphic MAC. Unlike traditional MAC schemes, such as HMAC, by using Combine, the tag of a data block can be generated by performing computation on existing tags of related blocks, and without the need of the secret key. This ensures that multiple blocks can be audit at the bandwidth and verification computation cost of a single block.

- **Verify($k$, id, e, t):**
  - Input: A secret key, $k$, the identifier, id, of the file, a block, $e \in \mathbb{F}_q^{n+m}$, and its tag, $t$.
  - Output: 0 (reject) or 1 (accept).

  Also, the scheme must satisfy the following correctness requirement:

  Let $t = \text{Combine}((e_1, t_1, \alpha_1), \ldots, (e_t, t_e, \alpha_e))$, then Verify $\left(k, id, \sum_{i=1}^t \alpha_i e_i, t\right) = 1$.

**Attack Game 1.** We consider the following attack game for a homomorphic MAC scheme $\mathcal{T}$. The game involves a challenger $C$ and an adversary $A$ as follows:

- **Setup.** $C$ generates a random key $k \overset{\$}{\leftarrow} \mathcal{K}$. 

• Queries. A adaptively queries C, where each query is of the form \( (id, y) \). For each query, C replies to A with the corresponding tag \( t \leftarrow \text{Mac}(k, id, y) \).
• Output. A eventually outputs a tuple \((id^*, y^*, t^*)\).

Up to the time \( A \) outputs, it has queried \( C \) multiple times. Let \( l \) denote the number of times \( A \) queried \( C \) using \( id^* \) and get tags for \( l \) vectors, \( y_1^*, \ldots, y_l^* \) of these queries. We consider that the adversary wins the security game if and only if

\[
\left(y_1^{(n+1)}, \ldots, y_l^{(n+m)}\right) \neq 0 \quad (\text{trivial forge otherwise}),
\]

Verify\((k, id^*, y^*, t^*) = 1\), and

\[y^* \notin \text{span}(y_1^*, \ldots, y_l^*).\]

Let \( \text{Adv}[A, T] \) denote the probability that \( A \) wins the above attack game. We define a secure homomorphic MAC scheme as follows:

**Definition 1.** A \((q, n, m)\) homomorphic MAC scheme \( T \) is secure if for all probabilistic polynomial-time adversaries \( A, \text{Adv}[A, T] \) is negligible (in the security parameter \( \lambda \)).

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

- **Mac**:\((k, id, e) \rightarrow t\): The MAC tag \( t \in \mathbb{F}_q \) of a source block or encoded block, denoted by \( e \in \mathbb{F}_q^{n+m} \), under key \( k \), can be computed by the following steps:

  \[
r \leftarrow (F_1(k, id, 1), \ldots, F_1(k, id, n + m)).
  \]

  \[
t \leftarrow e \cdot r \in \mathbb{F}_q.
  \]

- **Combine**:\((e_1, t_1, \alpha_1), \ldots, (e_\ell, t_\ell, \alpha_\ell) \rightarrow t\): The tag \( t \in \mathbb{F}_q \) of \( e \in \mathbb{F}_q^{n+m} \) is computed as follows:

  \[
t \leftarrow \sum_{i=1}^\ell \alpha_i t_i \in \mathbb{F}_q.
  \]

- **Verify**:\((k, id, e, t) \rightarrow \{0, 1\}\): To verify if \( t \) is a valid tag of \( e \) under key \( k \), we do the following:

  \[
r \leftarrow (F_1(k, id, 1), \ldots, F_1(k, id, n + m)).
  \]

  \[
t' \leftarrow e \cdot r.
  \]

  If \( t' = t \), output 1 (accept); otherwise, output 0 (reject).

**Lemma 1 (Theorem 1 [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.

Loosely speaking, a PRF is secure if it is probabilistically indistinguishable from a truly random function. The formal definition of PRFs can be found in Section 3.6 of [47]. We refer the reader to our previous work [27] for the security proof of SpaceMac in a general setting. Here we will provide an extended 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, \( e' \), is not a linear combination of the source blocks, then the storage node can only forge a valid MAC tag for \( e' \) with probability \( \frac{1}{q} \). The security when using \( \ell \) tags is improved to \( \frac{1}{2^\ell} \).

\[D. \ The \ Random \ Linear \ Encryption: \ NCrypt\]

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.

Below, we present the construction of NCrypt and show that it is a secure encryption against the Chosen-Plaintext Attack (CPA-secure). Loosely speaking, in the chosen-plaintext attack, the adversary can acquire encryptions of multiple file blocks, chosen adaptively at its will. A CPA-secure encryption ensures that the adversary, even when given this ability, should not be able to distinguish the encryption of two arbitrary blocks. A formal definition of CPA-secure encryptions can be found elsewhere [47].

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

- **Setup**:\((k, r) \rightarrow (p_1, \cdots, p_{n-1})\): This algorithm is run by the user to setup the encryption scheme. It takes as input a secret key \( k \) and a vector \( r \neq 0 \in \mathbb{F}_q^{m-2} \). 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:

  \[
  \text{Compute } p_i \leftarrow (F_2(k, i, 1), \cdots, F_2(k, i, n-2)) \in \mathbb{F}_q^{m-2}, \text{ for } i \in [1, n-1].
  \]

  \[
  \text{Compute } p_i \leftarrow r \cdot p_i \in \mathbb{F}_q, \text{ for } i \in [1, n-1].
  \]

- **Enc**:\((e, (p_1, \cdots, p_{n-1})) \rightarrow \langle \bar{e}, (r, p) \rangle\): This algorithm is run by the storage node to encrypt the first \( n-2 \) 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, \( e \), and the auxiliary elements, \( p_1, \cdots, p_{n-1} \). It computes the encryption, \( \langle \bar{e}, (r, p) \rangle \), of \( e \) as follows:

  \[
  \begin{align*}
  &\text{Compute } p_i, i \in [1, n-1], \text{ using key } k \text{ as in Setup}. \\
  &\text{Choose } r \text{ uniformly at random: } r \leftarrow \{0, 1\}^\lambda. \\
  &\text{Compute the masking coefficients: } \beta_i \leftarrow F_3(k, r, i) \in \mathbb{F}_q^m, \text{ for } i \in [1, n-1]. \\
  &\text{Compute the masking vector: } \bar{m} \leftarrow \sum_{i=1}^{n-1} \beta_i p_i \in \mathbb{F}_q^{m-2}. \\
  &\text{Compute } \bar{e} \leftarrow e + \bar{m} \in \mathbb{F}_q^{m-2}. \\
  &\text{Compute } p \leftarrow \sum_{i=1}^{n-1} \beta_i p_i \in \mathbb{F}_q.
  \end{align*}
  \]

In essence, the data is masked with a randomly chosen vector \( \bar{m} \). 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.

In particular, the 2 random padded elements is to control the number of equations in the system of equations \( \Phi_1 \) and \( \Phi_2 \) 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 \( p \)).
messages of length is a fixed-length private-key encryption scheme for NCrypt

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 (CPA-secure).

Proof: Intuitively, the security of NCrypt holds because $m$ looks completely random to an adversary who observes a ciphertext $(e, (r, p))$ since it is computationally difficult for the adversary to compute $\beta_i$'s without knowing the secret key $k$.

The proof follows a technique used to prove the security of Construction 3.24 in the book by Katz and Lindell [47]. We follow the standard notation [47]. Denote the CPA security experiment of an encryption scheme $\Gamma = (\text{Setup}, \text{Enc}, \text{Dec})$ and an adversary $A$ by $\text{PrivK}^\text{cPa}_{A, \Gamma} = 1$, the game is as follows:

- A key $k$ is chosen uniformly at random from $\{0, 1\}^\lambda$.
- The adversary $A$ is given $R, p_1, \ldots, p_{n-1}$, and oracle access to $\text{Enc}_K$. $A$ outputs a pair of messages $\epsilon_0$ and $\epsilon_1$, both are in $\mathbb{F}_q^{n-2}$.
- A random bit $b \leftarrow \{0, 1\}$ is chosen, and then a ciphertext $c \leftarrow \text{Enc}(k, \epsilon_b, (p_1, \ldots, p_{n-1}))$ is computed and given to $A$. We call $c$ the challenge ciphertext.
- The adversary $A$ continues to have oracle access to $\text{Enc}_K$, and outputs a bit $b'$.
- The output of the experiment is defined to be 1 if $b' = b$, and 0 otherwise. In case $\text{PrivK}^\text{cPa}_{A, \Gamma} = 1$, we say that $A$ succeeded.

Note that the security experiment we consider here is identical to the standard CPA experiment [47], except that the adversary has additional information about additional values used in the encryption scheme, including $R, p_1, \ldots, p_{n-1}$.

Let $\Gamma_1$ be an encryption scheme that is exactly the same as $\Gamma$ except that a truly random function $f_2$ is used in place of $F_2$. Let $\text{Adv}[B, F_2]$ be the probability of an adversary $B$ with similar runtime to $A$ 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 $\text{Adv}[B, F_2]$ is negligible in $\lambda$ and it can be shown that (details are provided in the proof of Construction 3.24 in the book by Katz and Lindell [47])

$$\text{Adv}[B, F_2] = |\text{Pr}[\text{PrivK}^\text{cPa}_{A, \Gamma} = 1] - \text{Pr}[\text{PrivK}^\text{cPa}_{A, \Gamma'} = 1]|. \quad (1)$$

Similarly, let $\Gamma_2$ be an encryption scheme that is exactly the same as $\Gamma_1$ except that a truly random function $f_3$ is used in place of $F_3$. Let $\text{Adv}[C, F_3]$ be the probability of an adversary $C$ with similar runtime to $A$ winning the PRF security game. Similar to the above, by the security of PRF, we have that $\text{Adv}[B, F_3]$ is negligible in $\lambda$

$$\text{Adv}[C, F_3] = |\text{Pr}[\text{PrivK}^\text{cPa}_{A, \Gamma_1} = 1] - \text{Pr}[\text{PrivK}^\text{cPa}_{A, \Gamma_2} = 1]|. \quad (2)$$

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

$$\text{Pr}[\text{PrivK}^\text{cPa}_{A, \Gamma_2} = 1] \leq \frac{1}{2} + \frac{g(\lambda)}{2^\lambda}. \quad (3)$$

Let $r_c$ denote the random string used when generating the challenge ciphertext, which is of the form $(e, (r_c, p))$ (by encrypting $\epsilon_0$). There are two cases:

(a) $r_c$ was never used by the oracle in the encryption algorithm to produce ciphertext to answer any of $A$'s queries: In the following, we will show that each element of any plaintext $\bar{e}$ is masked with a uniformly random value, thus the adversary will not be able to tell which message ($\epsilon_0$ or $\epsilon_1$) was encrypted, as in the case of one-time pad.

Parse $\bar{e}$ as $(e^{(1)}, \ldots, e^{(n-2)})$, $\bar{m}$ as $(m^{(1)}, \ldots, m^{(n-2)})$, and $\bar{p}$ as $(p^{(1)}, \ldots, p^{(n-2)})$. From a ciphertext returned from an oracle query of $\bar{e}$, the adversary can construct the following system of equations $\Phi_1$ by subtracting the query plaintext from the ciphertext:

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

(1)

(2)

(3)

Note that $p_j^{(j)}$ are not all zeros w.h.p. since they are chosen uniformly at random from $\mathbb{F}_q$ by $f_2$. Let $\beta_i$ be unknowns, $i \in [1, n-1]$. The above system of $n - 1$ linear equations is consistent regardless of the values of $m_j^{(j)}$'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 $m_j^{(j)}, j \neq w, j \in [1, n-2]$, are fixed. Then $m_w^{(w)}$ still can take any value in $\mathbb{F}_q$ equally likely because (i) for any value of $m_w^{(w)}$, there is the same number of solutions, which is $q^{n-1-s}$, and (ii) $\beta_j$ 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, $m_w^{(w)}$, independent of other masking elements $m_j^{(j)}, j \neq w, j \in [1, n-2]$. Therefore, the probability that $A$ outputs $b' = b$ is exactly $1/2$, as in the case of the one-time pad.

(b) $r_c$ was used by the oracle to answer at least one of $A$'s queries: In this case, $A$ may easily determine which of its messages was encrypted. This is because whenever the oracle returns a ciphertext $(e, (r, p))$, it learns the masking vector $\bar{m}$ associated with $r$ as $\bar{m} = \bar{e} - \bar{c}$. Thus, by leveraging the corresponding $\bar{m}$ of $r_c$, the adversary can tell if $\epsilon_0$ or $\epsilon_1$ was encrypted by actually decrypting the challenge response. Since $A$ 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

$$\text{Pr}[\text{PrivK}^\text{cPa}_{A, \Gamma} = 1] \leq \frac{1}{2} + \frac{g(\lambda)}{2^\lambda} + \epsilon(\lambda),$$

where $\epsilon$ is a cryptographically negligible function in $\lambda$. 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 $b_i$, $i \in [1, m]$.
- The user runs KeyGen to generate MAC verification key, $k_e$, and encryption key, $k_c$:
  \[ \overline{c} = \text{KeyGen}(1^\lambda) \rightarrow (k_e, k_c); \ k_e \leftarrow \{0, 1\}^\lambda, k_c \leftarrow \{0, 1\}^\lambda. \]
- The user then setups the encryption scheme by computing the auxiliary elements, $p_1, \ldots, p_{n-1}$:
  \[ \bar{r} = (F_1(k_e, 1), \ldots, F_1(k_e, n - 2)). \]
- The user then computes tags for the blocks to be sent to the storage node. An outsourced block $\bar{e}$ could be either a source block or an encoded block. TagGen implementation is as follows:
  \[ \bar{t}_j = \text{TagGen}(k_e, \bar{e}) = \text{Mac}(k_e, \bar{e}). \]
- Finally, the user sends the blocks, $e_1, \ldots, e_M$, their tags, $t_{e_1}, \ldots, t_{e_M}$, the auxiliary elements, $p_1, \ldots, p_{n-1}$, and the encryption key, $k_c$, to the storage node. The user also sends the coding coefficients, $\text{aug}(e_1), \ldots, \text{aug}(e_M)$, and the MAC key, $k_e$, to the TPA. We assume that the user uses private and authentic channels to send $k_e$ and $k_c$.
- The user then keeps the coding coefficients and the keys but deletes 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, $I \subseteq [1, M]$, and chooses the coefficients for these blocks uniformly at random: $\alpha_i \leftarrow \mathbb{F}_q$, $i \in I$. The challenge includes the indexes of the blocks and their corresponding coefficients:
  \[ \text{chal} = \{(i, \alpha_i) \mid i \in I\}. \]
- GenProof run by the storage node to generate the proof of storage, $V$, is implemented as follows:
  \[ \text{Compute the aggregated block: } \hat{e} = \sum_{i \in I} \alpha_i \bar{e}_i, \text{ Parse } \hat{e} \text{ as } (\hat{e}, e^{(n-1)}, e^{(n)}). \]
  \[ \text{Compute the aggregated tag: } t = \sum_{i \in I} \alpha_i t_{e_i}. \]

3Exchanging 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.

- Encrypt the response block: $\langle \hat{c}, (r_1, r_2) \rangle \leftarrow \text{Enc}(k_e, \hat{e}, (p_1, \ldots, p_{n-1}))$. The node then sends $V = ((\hat{c}, (r_1, r_2)), e^{(n-1)}, e^{(n)}, t)$ back to the TPA.
- VerifyProof run by the TPA to verify the proof $V$ is implemented as follows:
  \[ \text{Compute coefficients of } e: \text{aug}(e) = \sum_{i \in I} \alpha_i \text{aug}(e_i). \]
  \[ \text{Let } c = (\hat{c} | e^{(n-1)} | e^{(n)} | \text{aug}(e)), \text{ where } "|" \text{ denotes augmentation.} \]
  \[ \text{Return result of Verify}(k_v, c, t + p). \]

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 $r = (F_1(k_v, 1), \ldots, F_1(k_v, n + m))$. Note that

\[ c = (\hat{c} | e^{(n-1)} | e^{(n)} | \text{aug}(e)) = (\hat{c} + \overline{\hat{m}} | e^{(n-1)} | e^{(n)} | \text{aug}(e)) = e + (\overline{\hat{m}} | 0, \ldots, 0). \]

Thus, in the Verify, \[ t' = c \cdot r = e \cdot r + \overline{\hat{m}} \cdot \bar{r} = t + \sum_{i=1}^{n-1} \beta_i p_i = t + p. \]

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, $N_i, i = 1, \ldots, P$, recall that it needs to send $Q$ encoded repair blocks to the new node. Let $(e_{i,1}, \ldots, e_{i,M})$ be the encoded blocks currently stored on $N_i$. For $j = 1, \ldots, Q$, the user sends a set of repair coding coefficients $\gamma_{i,j,k}$ to $N_j$. This node then uses these coefficients to compute the repair blocks, $g_{i,j,k} = \sum_{k=1}^{M} \gamma_{i,j,k} e_{i,k}$, to send to the new node. The new node will receive $P \times Q$ repair blocks, $g_{i,j,k}$, from the healthy nodes. It uses them to reconstruct the encoded blocks, $h_{i,1}, \ldots, h_{i,M}$, that it needs to store. For $k = 1, \ldots, M$, the user sends a set of $P \times Q$ reconstruction coding coefficients, $(\theta_{i,j,k} | \cdots | p_{i,j,k})$, to the new node to instruct its reconstruction. The new node
then reconstructs $h_k = \sum_{i=1}^{P} \sum_{j=1}^{Q} \theta_{i,j,k} g_{i,j}$. Note that the coding coefficients $\gamma$’s and $\theta$’s are dependent on the repairing scheme.

Using NC-Audit, the verification tags of the newly constructed blocks, $h_k$, 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, $g_{i,j}$, that they send to the new node, where the tags of $g_{i,j}$ can be computed using the Combine algorithm of SpaceMac on the tags of $e_{i,k}$. The new node then can also use Combine on the tags of $g_{i,j}$ to generate tags of $h_k$. Finally, the user sends the coding coefficients of the coded blocks at the newly constructed node, $\text{aug}(h_k)$ (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., $e_{i,k}$, and the user also does not need to compute the tags, i.e., runs Mac on $h_k$. 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 outside 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].

**Single Storage Provider.** As mentioned previously in Section III-A, in the scenario where all the storage nodes are provided by the same storage provider, e.g., Amazon, this storage provider could assume the repair responsibility itself. In this case, we assume that the repair coding scheme is known by the TPA so that it can correctly audit the newly constructed nodes. (The scheme could be shared by the user or publicly available.)

To see why this assumption is necessary, recall that in order to audit a node, the TPA needs the coding coefficients of the encoded blocks stored at the node. However, as in our threat model, the storage provider has incentives to delete rarely accessed data or hide data corruption; thus, in the repair process, we cannot rely on it to send the correct coefficients of the blocks stored at the new node to the TPA. This is because it may send incorrect or duplicated coefficients to hide missing or corrupted data blocks. Lastly, if the storage node does not follow the predefined repairing scheme, the verification performed by the TPA on the blocks at the new node will fail due to mismatched coding coefficients.

**VI. Security Analysis**

**A. Data Possession Guarantee**

In this work, we consider the security of SpaceMac in a slightly different security game from Attack Game 1 (previously described in Section IV-C), 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 number of queries the adversary can make is at most $m$. This implies that any space spanned by the queried vectors of the same id has dimension at most $m$.

This Attack Game 2 is stricter but better fits our application: since the dimension of any source space $\Pi$ is only $m$, the adversary must only learn tags of vectors in spaces having dimensions at most $m$. In this system, this means that the security of SpaceMac holds regardless of the number of tags of multiple files that the storage node already learned.

When using SpaceMac in NC-Audit, some information about the vector $r$ in the SpaceMac construction is available to the adversary. In particular, the storage node knows the following $n-1$ equations: $p_i \cdot r = p_i$, $i \in [1, n-1]$. The following theorem states that even when these $n-1$ equations are exposed, SpaceMac is still a secure homomorphic MAC, 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 $A$ knows any $n-1$ linearly independent vectors, $\overline{p}_1, \cdots, \overline{p}_{n-1}$, and any $n-1$ constants, $p_1, \cdots, p_{n-1}$, such that $p_i \cdot r = p_i$, where $r$ is used in the construction of SpaceMac. The probability that $A$ wins the SpaceMac security game, denoted by $\text{Adv}[A, \text{SpaceMac}]$, is at most

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

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

**Proof:**

Let Game 0 be identical to Attack Game 2, and let $W_0$ denote the event that $A$ wins Game 0. Hence,

$$\Pr[W_0] = \text{Adv}[A, \text{SpaceMac}]. \quad (4)$$

Let Game 1 be identical to Game 0 except that the challenger $C$ computes $r \leftarrow (r_1, \cdots, r_{n+m})$, where $r_j, j \in [1, n+m]$, is chosen uniformly at random from $F_q^R$, where $R$ is chosen uniformly at random from $F_q$ instead of $r_j \leftarrow F(k, \text{id}, j)$, and everything else remains the same. And let $W_1$ denote the event that $A$ wins Game 1. Then, there exists a PRF adversary $B$ such that

$$|\Pr[W_0] - \Pr[W_1]| = \text{PRF-Adv}[B, F]. \quad (5)$$

The complete challenger in Game 1 works as follows:

**Queries.** $A$ adaptively queries $C$, where each query is of the form $\text{id} \cdot y$. If $\text{id}$ is already used in $m$ previous queries, $C$ discards the query. Otherwise, $C$ computes $r^\text{id} := (r^\text{id}_1, \cdots, r^\text{id}_{n+m})$, where $r^\text{id}_j \in F_q, j \in [1, n+m]$, and then replies to $A$ with $t := y \cdot r^\text{id}$. 

**Output.** $A$ eventually outputs a tuple $(\text{id}^*, y^*, t^*)$. To determine if $A$ wins, when $(y^{(n+1)}_1, \cdots, y^{(n+m)}_*) \neq 0$, if $\text{id}$ is
never used in any query before, compute \( r^{id^*} \) similarly to the above; denote \( r^{id^*} \) simply by \( r^* \).

Let \( l \) denote the number of times \( A \) queried \( C \) using id* and get tags for \( l \) vectors, \( y_1^*, \ldots, y_l^* \), of these queries. The adversary wins the game, i.e., event \( W_1 \) happens, if and only if

\[
\begin{align*}
t^* &= y^* \cdot r^*, \quad \text{and} \\
y^* &\not\in \text{span}(y_1^*, \ldots, y_l^*). 
\end{align*}
\]

(6)

(7)

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

- When \( T \) happens: in this case, \( r^* \) was never used to compute any response before. Because \( r^* \)'s of \( r^* \) are indistinguishable from random values and \((y_{(n+1)}^*, \ldots, y_{(n+m)}^*) \neq 0\), the right hand side of equation (6) is a completely random value in \( \mathbb{F}_q \). Thus,

\[
\Pr[W_1 \land T] = \frac{1}{q} \Pr[T].
\]

(8)

- When \( T \) does not happen: we proceed by showing that for a fixed \( y^* \), \( t^* \) 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 \( \Phi_2 \):

\[
\begin{align*}
\hat{p}_1 \cdot r^* &= p_1 \\
\vdots \\
\hat{p}_{n-1} \cdot r^* &= p_{n-1} \\
y_1^* \cdot r^* &= t_1^* \\
\vdots \\
y_l^* \cdot r^* &= t_l^* \\
y^* \cdot r^* &= t^* 
\end{align*}
\]

(\( \Phi_2 \))

Let the elements \( r_j^*, j \in [1, n + m] \), of \( r^* \) 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 \( y^* \), 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) \( r_j^* \)'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,

\[
\Pr[W_1 \land \neg T] = \frac{1}{q} \Pr[\neg T].
\]

(9)

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

\[
\Pr[W_1] = \Pr[W_1 \land T] + \Pr[W_1 \land \neg T] = \frac{1}{q}.
\]

(10)

Equations (4), (5), and (10) together prove the theorem. We now are ready to prove the data possession guarantee of NC-Audit.

Let \( q \) denote the probability that the storage node can guess correctly a missing or corrupted element of a file block. In other words, for any \( j \in [n + m] \) and any encoded block \( e \), if \( e^{(j)} \) of \( e \) is corrupted or missing, the storage node can recover \( e^{(j)} \) with probability at most \( g \). For instance, if an element of a file block could take any value in \( \mathbb{F}_q \) equally likely, then \( g = \frac{1}{q} \). We assume that \( g \) is low, in the order of \( \frac{1}{q} \).

Lemma 5. With probability at least \( 1 - g - \frac{1}{q} \), 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 contrapositive, i.e., if there are corrupted or missing blocks, the node will fail the check w.h.p.

Case (a) - The storage node is able to compute a correct response block even when there is some parts of some blocks are missing or corrupted.

Denote the correct, unencrypted aggregated block by \( e \), i.e., \( e = \sum_{i \in \mathcal{I}} a_i e_i \), where \( \mathcal{I} \) is the set of indexes of blocks to be audited. Let \( u, u \in \mathcal{I} \), be the index of a corrupted or missing block. If the block is corrupted, i.e., at least one of its element is incorrect, let \( v \) be the index of that element. If the block is missing, \( v \) could take any value in \( [1, n + m] \); let \( v = 1 \). Note that the \( v \)th element of the correct block \( e \) is computed as:

\[
e^{(v)} = \sum_{i \in \mathcal{I}} a_i e_i^{(v)} = \sum_{i \in \mathcal{I}, i \neq u} a_i e_i^{(v)} + a_u e_u^{(v)}.
\]

Without loss of generality, assume that the storage node knows all values of \( e_i^{(v)} \)'s except for \( e_u^{(v)} \). Denote the response block actually computed by the storage node by \( a \). Then,

\[
\Pr[a = e] \leq \Pr[a^{(v)} = e^{(v)}] \leq g,
\]

where the last inequality comes from the fact that \( a_u \) is chosen uniformly at random from \( \mathbb{F}_q \).

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 \( a \in \Pi \) 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 \( a \), it can provide the correct encryption to pass the check. Meanwhile, if the node does not know a valid tag of \( a \), its chance of forging a valid tag for the cipher text \( e \) 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 \( g + \frac{1}{q} \).

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 previous theoretical framework [19], which was derived from the earlier work [15], [16].

Lemma 6. Let \( p \) be the probability that the storage node responds correctly to a challenge, and let \( \gamma = 1 - p \). By performing \( \gamma < q \), challenge-response interactions with the storage node, the user can extract an encoded block \( e \) stored at the node with probability at least \( (1 - \gamma^2)(1 - \frac{1}{q}) \).

Proof: Consider the following simple extraction algorithm: the user challenges the storage node \( \gamma \) times, where each challenge consists of the index of just \( e \) and a different...
coefficient for $\epsilon$. The user runs the VerifyProof algorithm on each response. And as soon as there is a correct response, the user runs the Dec algorithm to decrypt and get $\epsilon$. The probability that there is at least one correct response out of $\gamma$ responses is $1 - e^{-\gamma}$. And by the security of SpaceMac, the probability that the correct response contains the correct block $\epsilon$ is $1 - \frac{\gamma}{2}$. □

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 block. Thanks to homomorphic 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, $\epsilon$; the random value, $r$, of size $\lambda$ (typically 80 bits [10]); the auxiliary tag, $\rho$, 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.

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, $\bar{e}$, (ii) the response tag, $t$, (iii) the masking vector, $\bar{m}$, and the auxiliary element, $p$. The total cost is dominated by the cost to compute $\bar{e}$ and $\bar{m}$, which 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 $\bar{m}$ 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 $\ell$ 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 the previous work [17], [23], we use $q = 2^8$, $\ell = 10$, and $\lambda = 80$ to provide 80-bit security. To get higher level of security, one could either increase $q$ or $\ell$ and must also increase the key size, $\lambda$, to ensure the corresponding level of security. We 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 [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: Wang-2009 [23], Wang-2010 [17], and Chen-2010 [13]. The reported numbers for Wang-2009 and Wang-2010 are taken from Wang-2010.
(The overhead of Wang-2009 is similar to an existing public-key based scheme [15].) We refer the reader to Wang-2010 for the detailed setup. We implement the checking scheme of Chen-2010 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 of Chen-2010 for the detailed description of this scheme. Each number reported for NC-Audit or Chen-2010 is the average of 100 runs on a computer with Intel Xeon 2.8 Ghz and 32 GB RAM. We note that among the three schemes under comparison, Chen-2010 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 Wang-2009 and Wang-2010. This is due to the fact that NC-Audit is symmetric-key based while Wang-2009 and Wang-2010 are public-key based and make heavily use of expensive bilinear mapping operations4. Chen-2010 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 Chen-2010 is much larger 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.

4Due to the fundamental difference: the use of expensive bilinear mapping operations in Wang-2009 and Wang-2010, we expect a similar gap (in order of magnitude) between the computational overhead of them and that of NC-Audit when we run them on the same hardware.

REFERENCES


Ahn Le is currently a research scientist at the California Institute for Telecommunications and Information Technology, Irvine. He also serves as the CTO of Shoelace Wireless, Inc., which he co-founded. He was an intern with Google in 2011 and a visiting postdoctoral researcher at the University of Southern California in 2012. He received a B.S. degree in Computer Science - Mathematics from the University of Manitoba, Canada in 2006, a M.Math degree in Computer Science from the University of Waterloo, Canada in 2008, and a Ph.D. degree in Computer Science from the University of California, Irvine in 2012. He received an NSF SBIR Award at Shoelace Wireless in 2013. His research interests include network coding, network security and privacy, cooperative mobile wireless networks, and machine learning.

Athena Markopoulou (S98-M02-SM13) is currently an Associate Professor in the Electrical and Computer Engineering department, at the University of California, Irvine. She has also held short-term/visiting positions at SprintLabs (2003), Arista Networks (2005) and IT University of Copenhagen (2013). She received the Diploma degree in Electrical and Computer Engineering from the National Technical University of Athens, Greece, in 1996, and the M.S. and Ph.D. degrees in Electrical Engineering from Stanford University in 1998 and 2003, respectively. She received the Henry Samueli School of Engineering Faculty Midcareer Award for Research (2014) and the NSF CAREER Award (2008). She has been an Associate Editor for IEEE/ACM Transactions on Networking (2013-2015).

Her research is in the broad area of computer networks and includes network measurement and modeling, mobile devices, online social networks, network security and privacy.

Alex Dimakis is an Assistant Professor at the Electrical and Computer Engineering department, University of Texas at Austin. From 2009 until 2012 he was with the Viterbi School of Engineering, University of Southern California. He received his Ph.D. in 2008 and M.S. degree in 2005 in electrical engineering and computer sciences from UC Berkeley and the Diploma degree from the National Technical University of Athens in 2003. During 2009 he was a CMI postdoctoral scholar at Caltech.

He received an NSF Career award in 2011, a Google faculty research award in 2012 and the Eli Jury dissertation award in 2008. He is the co-recipient of several best paper awards including the joint Information Theory and Communications Society Best Paper Award in 2012. He is currently serving as an associate editor for IEEE Signal Processing Letters and IEEE Transactions on Information Theory.

His research interests include information theory, coding theory and machine learning.