Secure Cloud Storage Meets with Secure Network Coding

Fei Chen
Department of Computer Science and Engineering, Chinese University of Hong Kong, N.T., Hong Kong
fchen@cse.cuhk.edu.hk

Tao Xiang
College of Computer Science, Chongqing University, Chongqing 400044, China
txiang@cqu.edu.cn

Yuanyuan Yang
Department of Electrical and Computer Engineering, Stony Brook University, Stony Brook, NY 11794, USA
yuanyuan.yang@stonybrook.edu

Sherman S. M. Chow
Department of Information Engineering, Chinese University of Hong Kong, N.T., Hong Kong
sherman@ie.cuhk.edu.hk

***Abstract*—This paper investigates the intrinsic relationship between secure cloud storage and secure network coding for the first time. Secure cloud storage was proposed only recently while secure network coding has been studied for more than ten years. We show in general how to construct a secure cloud storage protocol given any secure network coding protocol. Our construction suggests a systematic way to construct various secure cloud storage protocols. We also show that it is secure under a definition which captures the real world uses of the cloud storage. From our general construction, we propose a secure cloud storage protocol based on a recent secure network coding protocol. The protocol is the first publicly verifiable secure cloud storage protocol in the standard model, while the previous work is either not publicly verifiable, or security argument is only argued heuristically in the random oracle model. We also enhance the proposed protocol to support third-party public auditing, which has received considerable attention recently. Finally, we prototype the proposed protocol and evaluate its performance. Experimental results validate the effectiveness of the protocol.**

***Index Terms*—Cloud storage auditing, network coding, security, third-party public auditing**

I. Introduction

In this paper, we study the relationship between two different areas in the networking field: secure cloud storage and secure network coding, which seem not related to each other at the first glance. One is concerned with the problem of checking whether the data outsourced to the cloud remains intact as it was before being outsourced, while the other focuses on protecting a network code from being polluted during the routing. Solutions for the former problem, which include proof of retrievability (PoR) and provable data possession (PDP), were proposed only recently. On the other hand, the latter area has been examined for more than ten years. It will be helpful for both areas if we can connect them together.

For the first time, in this work we build a connection between the two areas. Specifically, we show how to construct a secure cloud storage protocol given any secure network coding protocol. The connection immediately implies that almost all previous protocols for secure network coding can be transformed for securing cloud storage. Notice that currently secure cloud storage protocols are designed in a rather ad hoc way and there are only few successful protocols. In other words, we can *automatically* have many secure cloud storage constructions from the existing secure network coding protocols. Based on this connection, we also propose several secure cloud storage protocols which satisfy different applications. We also hope our work can unveil the wisdom behind existing solutions of secure cloud storage to possibly contribute to and provide new perspective to the network coding community.

Secure cloud storage. This problem was first formally studied by Juels and Kaliski [1] and Ateniese et al. [2]. There are two main entities involved in these protocols: a user and a cloud storage provider (CSP). A user outsources the data to the cloud that promises to store the whole data. The user then confirms the data integrity by interacting with the cloud using a secure cloud storage protocol. The motivation of such integrity check is that the user's data could be modified without the acknowledgement of the user, due to various possible reasons, such as data loss caused by the CSP for its poor management, hardware/software failure, intentional discard of rarely accessed data (due to economic incentive), malicious data tampering by hacker, or external pressures (e.g., in countries with restricted freedom of speeches). Without a secure cloud storage protocol, the occurrence of these incidents may be hidden by the cloud and gone unnoticed.

The novelty of a secure cloud storage protocol is that the user can check the data integrity *without* possessing the actual data. Traditional techniques based on hash, message authentication codes (MACs), and digital signatures require the user to store the data locally. Some protocols (e.g., [3][4][5][6]) are publicly verifiable, i.e., anyone besides the user can verify the data integrity; while other protocols are privately verifiable since only the user with the secret key can check the data integrity. A more detailed survey is deferred to Section VIII.

Secure network coding. This problem was first proposed by Cai and Yeung [7] and Gkantsidis and Rodriguez [8]. Network coding is a routing paradigm different from the traditional store-and-forward method. Instead, a router in the network sends out encoded data packets, where the encoding is

The corresponding author is Tao Xiang (txiang@cqu.edu.cn). This research was supported by the National Natural Science Foundation of China (Nos. 61103211, 61373178), the Natural Science Foundation Project of CQ CSTC (No. cstc2013jcyjA40001), and the Fundamental Research Funds for the Central Universities (No. CDJZR13185501). Sherman Chow is supported by the Early Career Scheme and the Early Career Award of the Research Grants Council, Hong Kong SAR (CUHK 439713), and grants (4055018, 4930034) from Chinese University of Hong Kong.

a function of received data packets. Encoding can increase the network capacity for multicast tasks, and linear coding where a router sends out a linear combination of received data packets is proved to be sufficient to achieve the increased capacity [9][10]. This is especially useful in cooperative networks. However, this paradigm also incurs severe security concerns. If an encoded packet is modified illegally, this modification will quickly spread to the whole network since a router will encode all received packets. This attack is also known as pollution attack, which will make the data receivers fail to decode the data. Thus, when security is a critical concern the nodes need to check whether a data packet is polluted.

Essentially, the secure network coding problem is also a type of data integrity checking problem. After network coding was proposed, researchers have constructed many protocols for secure linear network coding. These solutions are also based on traditional data integrity techniques such as cryptographic hash functions [11], MACs [12] or digital signatures [13][14]. The intuition is that a codeword in the network needs to be checked whether it is modified illegally, i.e., the codeword needs to be authenticated. The challenge is that the packets in the network will be linearly combined and the new packets also need to be authenticated. If a protocol can enable a router to check the integrity of a combined codeword without any secret information, the protocol is said to be publicly verifiable. Indeed, a publicly verifiable protocol can remove the trust on intermediate routers, which is very useful in practice. To the best of the authors' knowledge, all current cryptographic solutions of secure network coding rely on some homomorphic property of the underlying cryptographic techniques. Readers may refer to Section VIII for more detailed related work.

OUR WORK. Firstly, we connect the areas of secure cloud storage and secure network coding for the first time; the connection will benefit both areas from each other. As a result, we obtain the first publicly verifiable secure cloud storage protocol which is secure in the standard model, i.e., without assuming that a hash function is a random oracle for ensuring the security of the protocol. Another contribution is that we extend the proposed protocol to support a more advanced functionality of third-party public auditing. This is important and has received considerable attention recently. Finally, we implement a prototype of the protocol and evaluate its performance, which makes a step toward deployment the protocol in practice. All the experimental results can be reproduced.

The rest of the paper proceeds as follows. Section II and Section III describe how we model the secure cloud storage problem and the secure network coding problem. The security models of both protocols are also discussed. Section IV presents our general construction of a secure cloud storage protocol from any secure network coding protocol. Its security analysis is also presented. Section V constructs a detailed secure cloud storage protocol based on a recent secure network coding protocol. Section VI enhances the protocol to support third-party public auditing. Experimental performance is discussed in Section VII. More detailed related work is discussed in Section VIII. Finally, Section IX concludes the paper by discussing some future work.

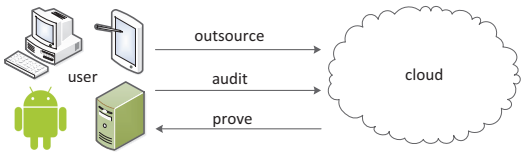

Fig. 1. A Secure Cloud Storage System

II. SECURE CLOUD STORAGE

In this section, we first model a secure cloud storage system in a simple way. Although the model is simple, it can be generalized to incorporate more properties as will be shown later. Then we present a high-level description of a protocol that ensures a secure cloud storage system. Last, we abstract the real-world usage of the protocol and then give a formal security definition.

A. System Model, Threat Model and Design Goals

We model a secure cloud storage system as shown in Fig. 1. There are two entities: user and cloud. In practice, a user could be an individual, a company, or an organization, using a PC or a mobile phone, etc.; a cloud could be any CSP, e.g., Amazon S3, Dropbox, Google Drive, etc. The user wants to *outsource* its data to the cloud. Later, the user wants to periodically performs an *audit* on the data integrity. The user can then check whether the proof is valid or not, meaning that the data remains intact, or obtaining an evidence that the data has been tampered and possibly some further action (which is out of the scope of the protocol) is required, such as legal action or data recovery. Normally, an error-correcting code could be applied to the data before outsourcing, and multiple different clouds could be adopted to ensure data recovery. To extend this model, a third-party auditor could be introduced [4] to shift the auditing task from the user to the third-party auditor.

As previous work on secure cloud storage, we assume that the cloud is malicious in the sense that, in order to cover some accidental data loss, it will try its best to prove that it is still storing the whole data. Data loss can hurt the reputation of the cloud and thus there is a strong incentive to fool the user.

A secure cloud storage system that enables a user to check the integrity of its data is expected to have the following properties: 1) Correct. If the cloud indeed stores the whole data of the user, the cloud can always prove to the user that the data remains intact. 2) Secure. If the user's data is changed, the user can detect such an abnormal event with high probability in the audit query even if the cloud tries to cover the event. 3) Efficient. The computation, storage and communication cost of both the user and the cloud should be as small as possible.

B. High-level Protocol Specification

It is clear that in order to verify whether the cloud lies to an audit query, the user needs to have some secret information on its side, which is computed according to a certain security level parameterizing the probability of successful cheating. A

secure cloud storage (SCS) protocol is thus a keyed protocol for the user to generate data to be outsourced and subsequently query for auditing. A privately verifiable scheme will also require the key for verification. Formally, it contains five efficient algorithms SCS = (KeyGen, Outsource, Audit, Prove, Verify) as follows

- $\mathsf{KeyGen}(\lambda) \to K$: On input a security parameter λ, the user runs this algorithm to generate a secret key K to enable audit and verification.
- $\mathsf{Outsource}(F; K) \to F'$: On input the data F to be outsourced, the user runs this algorithm to get the processed data F' using the secret key K. The processed data contains some authentication information of the outsourced data F and is then sent to the cloud.
- $\mathsf{Audit}(K) \to q$: The user runs this algorithm to generate an audit query q which will be sent to the cloud.
- $\mathsf{Prove}(q, F') \to \Gamma$: On input an audit query q, the cloud computes a proof Γ using the stored data F'.
- $\mathsf{Verify}(q, \Gamma; K) \to \delta$: On input an audit query q and the cloud's proof Γ, the user checks whether the cloud's proof is correct using the secret key K. The user outputs $\delta = 1$ if the proof is correct, else outputs $\delta = 0$.

C. Understanding Security

To understand the security of a system in a reasonable depth is not easy. There has been a lot of reports of security breaches of real world protocols [15][16]. The key to understand the security is to define the meaning of security exactly. Here we present a reasonable security definition of the secure cloud storage protocol step by step.

First, we need to understand the capability of a malicious cloud. In practice, the cloud has the processed data F'. Besides that, the cloud can see a lot of audit queries (and its proof responses). It is also reasonable that the cloud can know whether the user accepts a proof response or not. This is because if the user refuses to accept the proof, the user may sue the cloud or follow some other remedy actions; if the user accepts the proof, there will be no such actions. Another important question is that how many audit queries and verification results can be known to the cloud. In practice, this number should be small since the user will only send audit query periodically. If we allow the malicious cloud to see as many of such queries and verification results as possible, this malicious cloud is more powerful than that in practice. The rationale is that if a protocol can be secure under such a powerful adversary cloud, the protocol will definitely be secure in practice. Denote a malicious cloud by $\mathcal{A}$, a series of query, proof and verification result respectively by q_i, Γ_i, δ_i where $i = 1, 2, \ldots, \mathrm{poly}(\lambda)$, where $\mathrm{poly}(\lambda)$ is a fixed polynomial in the security parameter λ that bounds the maximal number of such audit/verification results the malicious cloud can see. We say a cloud cheats if it can find a query q^* and for this query the cloud sends back a Γ^*, where Γ^* is not computed using the whole processed data F', and the user accepts this proof. We denote the probability of such cheating behavior by $\Pr[\mathsf{Cheat}]$ and it is defined by

$$\Pr\left[\begin{array}{ll} \mathsf{KeyGen}(\lambda) \to K & \\ \mathsf{Outsource}(F; K) \to F' & \mathcal{A}(q_i, \Gamma_i, \delta_i) \\ \mathsf{Audit}(K) \to q_i & \text{outputs} \\ \mathsf{Prove}(q_i, F') \to \Gamma_i & (q^*, \Gamma^*) \\ \mathsf{Verify}(q_i, \Gamma_i; K) \to \delta_i & \text{and } \mathsf{Verify}(q^*, \\ i = 1, 2, \ldots, \mathrm{poly}(\lambda) & \Gamma^*; K) = 1 \end{array} : \right] \quad (1)$$

The left hand side denotes the capability of the malicious cloud and the right hand side denotes the cheating behavior. The probability is taken over the random choices of all algorithms and the malicious cloud.

Second, we need to understand the security intuition. If a cloud storage system is secure, then the user can confirm itself that the data on the cloud indeed remains the same as before being outsourced. One possible way to model this is to require the probability of a cheating behavior $\Pr[\mathsf{Cheat}]$ to be small. However, this is not sufficient since it is not directly related to whether the data remains intact at the cloud. It is better if the user can extract its data from the audit queries and the cloud's proof results. Denote the user by $\mathcal{U}$ and the probability of extracting the original data by $\Pr[\mathsf{Extract}]$. The probability $\Pr[\mathsf{Extract}]$ is defined by

$$\Pr\left[\begin{array}{ll} \mathsf{KeyGen}(\lambda) \to K & \\ \mathsf{Outsource}(F; K) \to F' & \mathcal{U}(q_i, \Gamma_i, \delta_i) \\ \mathsf{Audit}(K) \to q_i & \text{outputs } F^* \\ \mathsf{Prove}(q_i, F') \to \Gamma_i & \text{and} \\ \mathsf{Verify}(q_i, \Gamma_i; K) \to \delta_i & F^* = F \\ i = 1, 2, \ldots, \mathrm{poly}(\lambda) & \end{array} : \right] \quad (2)$$

The left hand side denotes the interactions of the user with the cloud. It is a kind of knowledge for the user. The user can issue as many queries as it wants since it owns the data. However, we require the total number of queries be bounded by a polynomial $\mathrm{poly}(\lambda)$ since it is more practical and the user only has bounded size data F. Even if a cloud can cheat, the protocol is still secure if the user can extract its whole data with a high probability. In other words, we require $\Pr[\mathsf{Extract}] \geq \Pr[\mathsf{Cheat}] - \mathrm{negl}(\lambda)$ where $\mathrm{negl}(\lambda)$ is a negligible function of λ. A function of λ is negligible if it is smaller than any inverse polynomial $\frac{1}{\mathrm{poly}(\lambda)}$ asymptotically. A good example to think about $\mathrm{negl}(\lambda)$ is $\frac{1}{2^\lambda}$. The intuition of security here is that if a malicious cloud can cheat with probability $\Pr[\mathsf{Cheat}]$, a user can extract its data with a probability at least roughly the same as $\Pr[\mathsf{Cheat}]$. This is a conservative intuition. Normally, in practice, we can achieve $\Pr[\mathsf{Cheat}]$ being negligible and $\Pr[\mathsf{Extract}]$ being approximately 1.

Summarizing the above discussion, we define the security of a secure cloud storage system as follows.

Definition 1. *A secure cloud storage system SCS = (KeyGen, Outsource, Audit, Prove, Verify) is secure if* $\Pr[\mathsf{Extract}] \geq \Pr[\mathsf{Cheat}] - \mathrm{negl}(\lambda)$ *where* $\Pr[\mathsf{Extract}]$ *and* $\Pr[\mathsf{Cheat}]$ *are defined in Equations (2) and (1), respectively.*

We remind that a stronger definition could require $\Pr[\mathsf{Cheat}]$ being negligible and $\Pr[\mathsf{Extract}]$ being as perfect as approximately 1. However, we choose the weaker definition since it is more general.

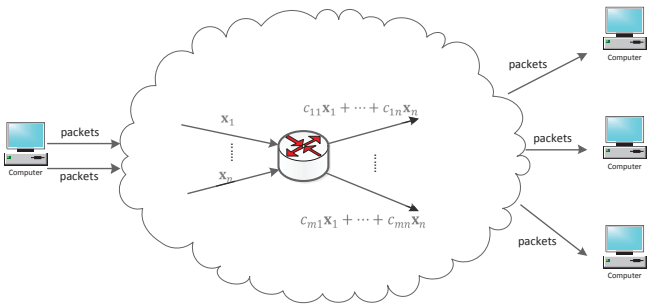

Fig. 2. A Typical Use of Network Coding

III. Secure Network Coding

In this section, we briefly introduce the concepts of secure network coding protocols. First, we discuss some basics about linear network coding and its use in practice. Then, we give a high-level description of a secure network coding protocol. At last, we give a security definition for such a protocol.

A. System Model

A typical system that employs network coding is shown in Fig. 2. There are three kinds of entities: *sender*, *router* and *receiver*. A sender has some data and wants to broadcast it to a group of receivers via the network. The sender divides the data into packets and sends a linear combination of them to the network. A router in the network also sends a linear combination of the received data packets to its next hops. When a receiver obtains sufficient encoded data packets, it can decode them to get the original data by solving a system of linear equations. To prevent a malicious router from modifying a packet, the sender attaches some authentication information with each data packet. When a router receives a packet, it first checks whether this packet is correct, i.e., not being modified. If yes, the router then combines the received packets and sends out the combined packet and the combined authentication information; if no, it simply discards the packet. The combined authentication information is computed according to a specific secure network coding protocol.

B. High-level Protocol Specification and Security Definition

We first introduce some notations. Denote the data to be sent by F and F is divided into m packets $\mathbf{v}_i$ where $i = 1, 2, \ldots, m$. Each data packet is a vector over some finite field $\mathbb{F}_p$, i.e., $\mathbf{v}_i \in \mathbb{F}_p^n$ where n is the length of the packet. Since data packets will be linearly combined during the transmission, the coefficients also need to be attached with the packets in the following way. Each $\mathbf{v}_i$ is attached with a unit vector $\mathbf{e}_i \in \mathbb{F}_p^m$ where only the i-th entry of $\mathbf{e}_i$ is 1 and the other entries are all 0. Denote the enhanced packet by $\mathbf{x}_i = [\mathbf{v}_i\ \mathbf{e}_i] \in \mathbb{F}_p^{n+m}$. When a router linearly combines a collection of packets, it will also update the coefficient vector in the output packet. For example, let $\mathbf{x}_j = [x_{j1}, \ldots, x_{jn}, c_{j1}, \ldots, c_{jm}]$ be a packet output by a router. Then, $[x_{j1}, \ldots, x_{jn}] = \sum_{k=1}^{m} c_{jk} \cdot \mathbf{v}_k$.

Abstracting the model in Fig. 2, a secure network coding (SNC) protocol contains four efficient algorithms SNC = (KeyGen, Auth, Combine, Verify) as follows

- $\mathsf{KeyGen}(\lambda) \rightarrow (SK, PK)$: On input a security parameter λ, the sender runs this algorithm to generate a secrete key SK and a public key PK to enable packet authentication.
- $\mathsf{Auth}(\mathbf{x}_i; SK) \rightarrow (\mathbf{x}_i, t_i)$: On input a packet $\mathbf{x}_i \in \mathbb{F}_p^{n+m}$ to be sent out in the network, the sender computes an authentication information t_i and sends out $(\mathbf{x}_i, t_i)$.
- $\mathsf{Combine}(\{\mathbf{u}_i, t_i\}_{i=1,\ldots,l}, \{c_1, \ldots, c_l\}) \rightarrow (\mathbf{w}, t)$: On receiving a group of packets $\mathbf{u}_i \in \mathbb{F}_p^{n+m}$ and their authentication information t_i's, a router runs this algorithm to generate a combined packet $\mathbf{w} \in \mathbb{F}_p^{n+m}$ with coefficients $\{c_1, \ldots, c_l\}$ and the combined authentication information t.
- $\mathsf{Verify}(\mathbf{w}, t) \rightarrow \delta$: On input a packet $\mathbf{w} \in \mathbb{F}_p^{n+m}$ and its authentication information t, a router or a receiver runs this algorithm to check whether a packet is modified maliciously. If the packet is correct, it outputs $\delta = 1$, else outputs $\delta = 0$.

To define the security, we need to understand the capability of a malicious router and the security intuition. The capability of a router is that it can see a lot of packets and their authentication information. The security intuition of a secure network coding protocol is that a malicious router cannot modify a packet illegally. If a router can find a forgery packet/authentication pair $(\mathbf{u}^* = [u_{j1}, \ldots, u_{jn}, c_{j1}, \ldots, c_{jm}], t^*)$ such that $[u_{j1}, \ldots, u_{jn}] \neq \sum_{k=1}^{m} c_{jk} \cdot \mathbf{v}_k$ and the Verify algorithm accepts this pair. Then, the protocol will be insecure. Let $\mathcal{A}$ be a malicious router and $\mathrm{Adv}[\lambda]$ be the probability of finding a forgery packet/authentication pair. Then, $\mathrm{Adv}[\lambda]$ can be defined by

$$\Pr\left[\begin{array}{ll} \mathsf{KeyGen}(\lambda) \rightarrow (SK, PK) & \\ \mathsf{Auth}(\mathbf{x}_i; SK) \rightarrow (\mathbf{x}_i, t_i) & \mathcal{A}(\mathbf{w}_j, t_j, \delta_j) \\ i = 1, 2, \ldots, m & \text{outputs} \\ \mathsf{Combine}(\{\mathbf{u}_i^{(j)}, t_i\}_{i=1,\ldots,l}, & : \quad (\mathbf{u}^*, t^*) \\ \{c_1^{(j)}, \ldots, c_l^{(j)}\}) \rightarrow (\mathbf{w}_j, t_j) & \text{and Verify} \\ \mathsf{Verify}(\mathbf{w}_j, t_j) \rightarrow \delta_j & \text{accepts it} \\ j = 1, 2, \ldots, \mathrm{poly}(\lambda) & \end{array}\right]$$

The left hand side shows the information $(\mathbf{w}_j, t_j, \delta_j)$ a malicious router could get in the network and the right hand side denotes the malicious behavior of a router. Similar to the security discussion of a secure cloud storage protocol, the security of a secure network coding protocol is defined as follows:

Definition 2. *A secure network coding protocol is said to be secure if* $\mathrm{Adv}[\lambda]$ *is negligible.*

We note that currently there has been many secure protocols [11][14][17] that employ the same semantics as shown here SNC = (KeyGen, Auth, Combine, Verify). Thus, we only focus on the high-level specification of a secure network coding protocol in the following discussion.

IV. Building Cloud Storage from Network Coding

This section shows how to construct a secure cloud storage protocol from a secure network coding protocol. The construction is based on some critical observations. We formally show that the general construction is secure with respect to Definition 1. The intuition of the security proof is simple; however, there are considerable technical details. Finally, we discuss our design choices and other possible constructions.

A. General Construction

Our goal is to construct a secure cloud storage protocol SCS = (KeyGen, Outsource, Audit, Prove, Verify) given a well-designed secure network coding protocol SNC = (KeyGen, Auth, Combine, Verify), i.e., the semantics of SNC is known, and we will work out the semantics for SCS. We first show the basic idea, then we give the detailed construction.

The basic idea lying behind the general construction is intuitive. We treat the user as a sender who wants to send the data to some receivers; however, we also treat the user as a data receiver or a router in the network. The cloud plays as a router in the network. When the user outsources the data, it first divides the data into packets which can be seen as a vector over some finite fields. Then, it authenticates the data packets by attaching some authentication information. The authenticated data is outsourced to the cloud. When the user sends an audit query to the cloud, we treat the cloud as a router that receives some data packets in the network. The indices of the data packets and the coefficients of the encoding are sent by the user as an audit query. The linearly encoded packet and its authentication information is then sent by the cloud back to the user as a proof. In this case, the user acts like a next-hop router or a data receiver. By checking if the data packet is valid, the user knows that whether the cloud is honest.

A Simple Construction. To implement the above idea, we need to divide the data F into packets $\mathbf{v}_i \in \mathbb{F}_p^n$ which is also attached with a coefficient vector $\mathbf{e}_i$. Then, we can take the data F as a collection of packets $\mathbf{x}_i = [\mathbf{v}_i \ \mathbf{e}_i] \in \mathbb{F}_p^{n+m}$. To outsource the data, the user generates the authentication information using SNC.Auth and outsources the processed file $F' = \{(\mathbf{x}_i, t_i)\}_{i=1,\ldots,m}$. To audit the data, the user asks the cloud to output a combined data packet and its authentication information. The user sends a collection of indices and coefficients $\{i_j, c_j\}_{j=1,\ldots,l}$ where l is the number of packets a router receives, i.e., taking the cloud as a router. For efficiency consideration, we set l as a constant, which does not depend on m. The cloud returns the combined data packet $(\mathbf{w}, t)$ which is output by $\mathsf{SNC.Combine}(\{\mathbf{u}_i, t_i\}_{i=1,\ldots,l}, \{c_1, \ldots, c_l\})$ where $\mathbf{u}_i = \mathbf{x}_{i_j}$. To verify whether the cloud cheats, the user can check the authentication information of $\mathbf{w}$ using $\mathsf{SNC.Verify}(\mathbf{w}, t)$.

An Optimized Construction. The above construction has a considerable storage cost for the cloud since the coefficient vector $\mathbf{e}_i$'s should also be stored. If the packet size n is small, the total number of packets m is as large as $\frac{|F|}{n \cdot |p|}$ where $|F|$ is the bit length of the data and $|p|$ is the bit length of the finite field order. The value of m could be much larger than n, which will incur a considerable additional storage cost. We observe that the user does not need to embed the coefficients in the data packets in our adaptation of a secure network coding protocol into our secure cloud storage protocol. The reason why the coefficients are needed to be embedded in the packet is that this helps a data receiver who does not know the coefficients to decode the data correctly. Now in our adaptation, the user acts as the data sender and it also acts as a data receiver or a router. Thus, the coefficients are chosen by the user itself and not required to be embedded. This is also useful for the security of the secure network coding protocol. The remaining part of the protocol is the same as the simple construction.

We summarize the secure cloud storage protocol SCS = (KeyGen, Outsource, Audit, Prove, Verify) by filling in the detailed semantics as follows.

- $\mathsf{KeyGen}(\lambda) \to K$: The user determines the finite field $\mathbb{F}_p$ where the network coding is done. The user also determines the packet size n and the total number of packets m. Then the user runs $\mathsf{SNC.KeyGen}(\lambda) \to (SK, PK)$. The key is $K = (SK, PK)$.
- $\mathsf{Outsource}(F; K) \to F'$: On input the data F to be outsourced, the user takes F as a collection of vectors $\{\mathbf{v}_i\}_{i=1,\ldots,m}$ in $\mathbb{F}_p^n$. Take each $\mathbf{v}_i$ as a codeword and then attach it with a coefficient vector $\mathbf{e}_i$ to get $\mathbf{x}_i = [\mathbf{v}_i \ \mathbf{e}_i] \in \mathbb{F}_p^{n+m}$. The user runs $\mathsf{SNC.Auth}(\mathbf{x}_i; SK) \to (\mathbf{x}_i, t_i)$ to get the authentication information. The user then outsources the processed data $F' = \{\mathbf{v}_i, t_i\}_{i=1,\ldots,m}$ to the cloud. Note that we outsource the $\mathbf{v}_i$'s but not the $\mathbf{x}_i$'s.
- $\mathsf{Audit}(K) \to q$: The user runs this algorithm to generate a collection of uniformly random numbers $\{i_j, c_j\}_{j=1,\ldots,l}$ where $1 \le i_j \le m$ and $c_j \in \mathbb{F}_p$. The user sends the query $q = \{i_j, c_j\}_{j=1,\ldots,l}$ to the cloud. Note that to achieve a good security level, the user will send multiple independent audit queries during one audit process.
- $\mathsf{Prove}(q, F') \to \Gamma$: On receiving an audit query $q = \{i_j, c_j\}_{j=1,\ldots,l}$ where l is the length of the query, the cloud augments $\mathbf{v}_{i_j}$ with the unit coefficient vector $\mathbf{e}_{i_j}$ to get a codeword for all j. The cloud runs $\mathsf{SNC.Combine}(\{\mathbf{u}_i, t_i\}_{i=1,\ldots,l}, \{c_1, \ldots, c_l\}) \to (\mathbf{w}, t)$. The cloud extracts the first n entries of $\mathbf{w}$ as a vector $\mathbf{y} \in \mathbb{F}_p^n$. The cloud sends back $(\mathbf{y}, t)$ as a proof of the corresponding query. Note that we have $\mathbf{y} = \sum_{j=1}^{l} c_j \cdot \mathbf{v}_{i_j}$.
- $\mathsf{Verify}(q, \Gamma; K) \to \delta$: On input an audit query $q = \{i_j, c_j\}_{j=1,\ldots,l}$, the cloud's proof $\Gamma = (\mathbf{y}, t)$, the user constructs a vector $\mathbf{w} \in \mathbb{F}_p^{n+m}$ such that the first n entries of $\mathbf{w}$ are the same as $\mathbf{y}$, the $(n + i_j)$-entry is c_j, and all other entries are 0. The user runs $\mathsf{SNC.Verify}(\mathbf{w}, t) \to \delta$ to get an answer δ. If $\delta = 1$, the cloud is honest; else, the cloud cheats.

B. Security Analysis

Now we show our general construction of a secure cloud storage protocol is secure under Definition 1 if a secure network protocol is used. The proof can be done in two steps. In Step I, we show a malicious cloud cannot cheat with a

high probability, i.e., $\Pr[\mathsf{Cheat}]$ as defined in Equation (1) is small. Step I is fairly easy since the underlying secure network coding protocol is secure. In Step II, we show the original data can be extracted by the user with a high probability and $\Pr[\mathsf{Extract}] \geq \Pr[\mathsf{Cheat}] - \mathrm{negl}(\lambda)$. Step II deals with some combinatorial and probabilistic argument on the rank of certain random matrix. We omit the detailed proof here and will show it in the full paper.

Theorem 3. *The general construction of SCS is secure if the underlying construction component SNC is secure.*

C. Further Discussion

DESIGN SPACE. There are several design choices in the general construction. First, when network coding is applied in practice, the data could be divided into multiple generations and each generation is transmitted using the network coding technique. However, the general construction of the secure cloud storage protocol treats the whole data as a whole generation. If multiple generations are adopted, we need to audit each generation to make sure that the user's data remains intact. Second, in network coding enabled systems, the packet size is much larger than the total number of packets, i.e., $n \gg m$. This helps to reduce the additional cost to transmit the coefficients in the network. However, in the secure cloud storage protocol, it is required that $n \ll m$ since we want the communication between the user and the cloud to be as little as possible. Let l be the length of an audit query and k be the total number of audit queries during one audit process. Then, the protocol cannot detect the cheating behavior with probability at most $(1 - \frac{l}{m})^k$, which is exponentially small with respect to k. In practice, we can set the length of the audit query l as a constant, e.g., 50 or 100. Then, the total number of audit queries can be set accordingly.

OTHER DIRECTIONS. It is also interesting to investigate whether a secure network coding protocol can be constructed in general from any secure cloud storage protocol. For example, we can use a keyed cryptographic hash function to compute the multiple digests of the outsourced data under different keys; in the audit phase, we can ask the cloud to return a digest under a secret key. This protocol cannot be easily used to construct a secure network coding protocol. The intrinsic reason is that a secure network coding protocol requires to support authentication of linearly combined packets while it is not for secure cloud storage. However, there do exist some secure cloud storage protocols that can be converted to secure network coding protocols [3]. It is an interesting future work to investigate whether a secure cloud storage protocol enhanced with some special property can be used to construct a general secure network coding protocol.

V. A NEW SECURE CLOUD STORAGE PROTOCOL

In this section, we construct a new secure cloud storage protocol, which is based on a recent secure network coding protocol [17]. Compared with previous protocols for secure cloud storage, the new protocol is the *first* publicly verifiable secure cloud storage protocol, which is secure without assuming the random oracle model. We also evaluate its theoretical performance and compare it with some recent secure cloud storage protocols.

A. Protocol Detail

We show the new protocol by filling all the semantics of a secure cloud storage protocol SCS = (KeyGen, Outsource, Audit, Prove, Verify). Details are as follows:

- $\mathsf{KeyGen}(\lambda) \to K$: The user generate two random primes p, q and sets $N = pq$. The user then generates a prime e whose length is larger than the message length by 1 bit. The network coding is done over the finite field $\mathbb{Z}_e$. Then the user determines the packet size n and the total number of packets m. The user also generates $g, g_1, \ldots, g_n, h_1, \ldots, h_m$ which are coprime to N. The secret key is $SK = (p, q)$ and the public key is $PK = (N, e, g, g_1, \ldots, g_n, h_1, \ldots, h_m)$. Denote the key by $K = (SK, PK)$.
- $\mathsf{Outsource}(F; K) \to F'$: On input the data F to be outsourced, the user divides F into a collection of vectors $\{\mathbf{v}_i = [v_{i1}, \ldots, v_{in}]\}_{i=1,\ldots,m}$ in $\mathbb{Z}_e^n$. For each $\mathbf{v}_i$, compute its authentication information as follows. First, generate a random integer $s \in \mathbb{Z}_e$ uniformly. Find an $x \in \mathbb{Z}_N$ such that $x^e = g^s \cdot (\prod_{j=1}^n g_j^{v_j}) \cdot h_i \bmod N$. Then the authentication information for $\mathbf{v}_i$ is $t_i = (s, x)$. The user then outsources the processed data $F' = \{\mathbf{v}_i, t_i\}_{i=1,\ldots,m}$ to the cloud.
- $\mathsf{Audit}(K) \to q$: The user runs this algorithm to generate a collection of numbers $\{i_j, c_j\}_{j=1,\ldots,l}$ where $1 \leq i_j \leq m$ and $c_j \in \mathbb{Z}_e$. The user sends the query $q = \{i_j, c_j\}_{j=1,\ldots,l}$ to the cloud.
- $\mathsf{Prove}(q, F') \to \Gamma$: On receiving an audit query $q = \{i_j, c_j\}_{j=1,\ldots,l}$ where l is the length of the query, the cloud first finds the authentication information (s_{i_j}, x_{i_j}) for each queried data blocks. The cloud then computes $s = \sum_{j=}^{l} c_j s_{i_j} \bmod e$ and $s' = (\sum_{j=}^{l} c_j s_{i_j} - s)/e$. The cloud augments $\mathbf{v}_{i_j}$ with the unit coefficient vector $\mathbf{e}_{i_j}$ to get a codeword $\mathbf{u}_{i_j}$ for all j. The cloud computes $\mathbf{w} = \sum_{j=1}^{l} c_j \mathbf{u}_{i_j} \bmod e$ and $\mathbf{w}' = (\sum_{j=}^{l} c_j \mathbf{u}_{i_j} - \mathbf{w})/e$. Then, the cloud computes

$$x = \frac{\prod_{j=1}^{l} x_{i_j}^{c_j}}{g^{s'} \prod_{j=1}^{n} g_j^{\mathbf{w}'_j} \prod_{j=1}^{m} h_j^{\mathbf{w}'_{n+j}}}.$$

  The cloud extracts the first n entries of $\mathbf{w}$ as a vector $\mathbf{y} \in \mathbb{Z}_e^n$ and the authentication information of it is $t = (s, x)$. The cloud sends back $\Gamma = (\mathbf{y}, t)$ as a proof of the corresponding query.
- $\mathsf{Verify}(q, \Gamma; K) \to \delta$: On input an audit query $q = \{i_j, c_j\}_{j=1,\ldots,l}$, the cloud's proof $\Gamma = (\mathbf{y}, t)$, the user constructs a vector $\mathbf{w} \in \mathbb{F}_p^{n+m}$ such that the first n entries of $\mathbf{w}$ are the same as $\mathbf{y}$, the $(n + i_j)$-entry is c_j, and all other entries are 0. The user checks whether $x^e = g^s \cdot (\prod_{j=1}^{n} g_j^{\mathbf{w}_j}) \cdot (\prod_{j=1}^{m} h_j^{\mathbf{w}_{n+j}}) \bmod N$. If they are

TABLE I
PERFORMANCE COMPARISON OF THE NEW PROTOCOL WITH RECENT PROTOCOLS: 'ROM' DENOTES THE RANDOM ORACLE MODEL; 'CRYPTO' DENOTES CRYPTOGRAPHIC OPERATIONS

	User			Cloud			
Protocols	Computation	Storage	Communication	Computation	Storage	Communication	Security models
This Work	$O(m \cdot n \cdot \mathsf{crypto})$	$O(\lambda)$	$O(l)$	$O(l \cdot n \cdot \mathsf{crypto})$	$O(\lvert F\rvert + \frac{\lvert F\rvert}{m})$	$O(n+\lambda)$	Standard
Yang and Jia [6]	$O(m \cdot n \cdot \mathsf{crypto})$	$O(\lambda)$	$O(l)$	$O(l \cdot n \cdot \mathsf{crypto})$	$O(\lvert F\rvert + \frac{\lvert F\rvert}{m})$	$O(n+\lambda)$	ROM
Xu and Chang [5]	$O(m \cdot n \cdot \mathsf{crypto})$	$O(\lambda)$	$O(l)$	$O(l \cdot n \cdot \mathsf{crypto} + \lambda)$	$O(\lvert F\rvert + \frac{\lvert F\rvert}{m})$	$O(1+\lambda)$	Standard
Wang et al. [4]	$O(\lvert F\rvert \cdot \mathsf{crypto})$	$O(\lambda)$	$O(l)$	$O(l \cdot n \cdot \mathsf{crypto})$	$O(2 \cdot \lvert F\rvert)$	$O(1+\lambda)$	ROM

equal, the cloud is honest and output $\delta = 1$; else, the cloud cheats and output $\delta = 0$.

Now we give an example choice of parameters for our new protocol to make our discussion more concrete. We can set $e = 257$, $n = 1024$ and p, q are 1024-bit primes. Then, a data block is composed of 1024 sub-block each with 8-bit data; that is the data block is 1KB. The total number of blocks is $m = \frac{|F|}{8n}$. In Section VII, we will evaluate the experimental performance of the new protocol with these specified parameters.

B. Analysis

Security of this protocol follows from Theorem 3 and the security of the underlying secure network coding protocol [17]. The performance of the new protocol is summarized in Table I. We also compare the performance of the new protocol with recent secure cloud storage protocols [6][5][4]. Note that we focus on asymptotic performance of protocols with respect to data size $|F|$, block size $|n|$, total number of blocks $|m|$, length of the audit query l and security parameter λ. Since these protocols are very different in the underlying construction components, we omit many technical details and only focus on the key influential factors.

We compare the new protocol with several recent protocols. Compared with the protocol proposed by Yang and Jia [6], the main difference is the security foundation. The protocol in this paper is secure in the standard model while the protocol in [6] is secure in the random oracle model. The protocol proposed by Xu and Chang [5] is better in the communication cost; the computation cost for the cloud is also a little higher. The protocol proposed by Wang et al. [4] performs better in the communication cost; it also takes a little more storage cost and it is secure in the random oracle model. Different protocols have different strength and weakness; and they could be used for different applications.

VI. A NEW THIRD-PARTY PUBLIC AUDITING PROTOCOL

In this section, we show how to enhance the proposed new protocol to obtain an advanced protocol [4] which supports third-party public auditing. Wang et al. [4] proposed a new secure cloud storage protocol which introduces a third-party auditor to help the user to audit the data. The third-party auditor may have more experience and knowledge than the user. This enhancement will also reduce the burden of the user. The setup of this protocol will be almost the same as we previously described. The only exception is that instead of the data user, the third-party auditor will challenge the cloud to check whether the data remains intact. In such an enhanced protocol, it is desired that the third-party auditor should not know the user's data, but at the same time the third-party auditor can indeed verify the integrity of the user's data in the cloud.

In fact, only one algorithm SCS.Prove in our protocol SCS = (KeyGen, Outsource, Audit, Prove, Verify) needs to be enhanced. The key idea is to blind the message. Details are as follows.

- $\mathsf{Prove}(q, F') \rightarrow \Gamma$: When outsourcing the data to the cloud, the user also sends some random data blocks $\mathbf{w}_i$ in $\mathbb{Z}_e^{n+m}$ whose coefficients are set to 0. When receiving a challenge query from the third party auditor, the cloud computes the result and proof Γ in the same way as before to obtain $(\mathbf{y}, t)$. Then the cloud uses the $\mathbf{w}_i$'s to randomize Γ by adding $\mathbf{y}$ with some random linear combination of $\mathbf{w}_i$'s. Note that the authentication of $\mathbf{y}$ and $\mathbf{w}_i$'s are known to the cloud, this randomization step is quite easy using the linear homomorphism property of the authentication algorithm as in the original SCS.Prove. The randomization will mask the data and thus the third-party auditor cannot get knowledge about the user's data from cloud's response. The proof is still true in this case and thus the third-party auditor can indeed check the integrity of the user's data.

With the enhanced Prove algorithm, we have constructed a new secure cloud storage protocol that supports third-party public auditing. Compared with the original protocol, the proposed new protocol does not need to assume the random oracle model. On the other hand, the enhancement also shows that our general construction is powerful. Instead of designing a protocol in an ad hoc way, we can design it in a more systematic way.

VII. PERFORMANCE EVALUATION

In this section, we present the experimental performance evaluation results of our detailed protocol in Section V. All the experimental results can be reproduced since we use public data sets [18] and we post our source code online [19].

The details of experiments are as follows. We first build a basic prototype of the protocol using Java 7.0. Then, we measure the performance of the protocol in terms of its storage cost, communication cost and computation cost. The experiments are done on a PC with an Intel i3 3.1G CPU and 4GB memory. We use the snapshots of the Wikipedia database

TABLE II
WIKIPEDIA DATA SET (* REFERS TO 'SIMPLEWIKI-20130608')

Benchmark	File Name	File Size
#1	*-oldimage.sql.gz	2.27KB
#2	*-pages-articles-multistream-index.txt.bz2	1.45MB
#3	*-stub-meta-current.xml.gz	23.5MB
#4	*-pages-meta-current.xml.bz2	121MB
#5	*-pages-meta-history.xml.7z	432MB

TABLE III
ADDITIONAL STORAGE AND COMMUNICATION COST: n DENOTES THE BLOCK SIZE AND m DENOTES THE TOTAL NUMBER OF BLOCKS

Benchmark	n	m	Storage	Communication
#1	1KB	3	0.8KB	376B
#2	1KB	1488	0.54MB	376B
#3	1KB	24089	8.68MB	376B
#4	1MB	122	0.04MB	376B
#5	1MB	433	0.16MB	376B

as our public test data sets [18], which are generated by the Wikimedia dump service. The data set information is shown in Table II. To authenticate the outsourced data, we choose p and q to be a 1024-bit large integer in the protocol.

STORAGE COST. For the user, it only needs to store the secret key; thus, the storage cost is two long integers p and q. For the cloud, it needs to store both the data and the authentication information. The additional storage cost of our protocol is due to the authentication information of data. The experimental result is shown in Table III. We can see that the storage cost depends on the total number of blocks m linearly. Thus, it is reasonable that we should increase the block size n with the increase of data size. The storage cost can be very small if the block size is well chosen. For example, the additional storage cost of test #5 is only 0.04% compared with the original data size. However, block size n cannot be too large in practice since this will result in high communication cost, which depends mainly on block size. Thus, we need to have a trade-off between storage cost and communication cost.

COMMUNICATION COST. For the user, the communication cost depends on the length of the audit query, which is a constant and thus trivial. For the cloud, the communication cost consists of two parts: one is the linear combination of the queried data blocks and the other is the authentication information. The former is the same as block size; the latter depends on the size of authentication information. The experimental result is also shown in Table III, which only focuses on the additional authentication cost since the block size is fixed. From Table III, the additional communication cost is small, which is due to the short size of the aggregated authentication information. Note that the size of authentication information is two long integers, which only depends on the security level of the protocol, but not the file size.

COMPUTATION COST. For the user and the cloud, the computation cost is composed of four parts, i.e., the time for outsourcing, auditing, proving and verifying the data. The experimental result is shown in Table IV. Outsourcing the data takes much longer time than other operations. The time

TABLE IV
COMPUTATION COST (THE TIME COST IS MEASURED IN MILLISECONDS.)

Benchmark	n	m	Outsource	Audit	Prove	Verify
#1	1KB	3	832	0.03	828	726
#2	1KB	1488	1119599	0.04	927	732
#3	1KB	24089	18409614	2.00	1009	768
#4	1MB	122	87137897	0.38	970688	738411
#5	1MB	433	325039357	4.20	976570	741242

cost grows higher when the data size becomes larger. In theory, it depends on data size, block size and total number of blocks. The maximal time is 90.3 hours for data set #5 of 432MB. This is very slow; however, this is one-time cost and it can be amortized in subsequent audit queries. To audit the data, it is pretty fast. It is simply the time to generate the random indices and their linear coefficients for the audit query. The time for proving data possession is much shorter than that of outsourcing. It depends on block size, length of the audit query and time to generate the aggregated authentication information. The maximal time is roughly 16 minutes for test #5, which is as expected. For verifying a proof, it takes shorter time than the proving process. The maximal time cost is about 12 minutes for test #5. It can also be found that the verification time is tightly related to block size and total number of blocks.

Finally, we remark that the computation cost is still not satisfactory in the current prototype. The experiments show that the protocol is feasible in practice; but the efficiency could be further improved. On the other hand, while some existing protocols which are only secure in the random oracle model appear to be more efficient, the stronger assumption may make the deployer to use a larger security parameter to cater for the weaker security guarantee, and hence the same cryptographic operation in those schemes may actually be slower than an instantiation of our scheme. Also, we believe that we have made a step forward in secure cloud storage and the understanding of the relationship between secure cloud storage and secure network coding. There is still room for optimizing our prototype. We leave it as a future work.

VIII. RELATED WORK

Cloud storage auditing was first formally studied by Juels and Kaliski [1] and Ateniese et al. [2]. Juels and Kaliski proposed a protocol called POR which can verify whether the cloud stores the user's whole data based on some random authentication information. One drawback is that auditing can only be done a finite number of time . The work of Ateniese et al. also address the cloud storage auditing problem by creating some authentication information which is related to the data. Later, researchers worked out more protocols. Shacham and Waters [3] proposed two protocols based on message authentication codes (MAC) and digital signatures. Wang et al. proposed an extension based on bilinear maps. Yang and Jia [6] also gave a similar protocol. Xu and Chang [5] proposed a secure cloud storage protocol based on a special commitment protocol. There is also some interesting work based on number-theoretic-related hash functions [20]. The

drawback is that there lacks a convincing security argument in the hash function based protocols. All the above protocols are designed in an ad hoc way; on the other hand, a general construction is investigated in this paper.

Network coding was first proposed by Ahlswede et al. [9] as a technique to increase the throughput of a multicast network. Its security issue was first studied by Cai and Yeung [7] and Gkantsidis and Rodriguez [8]. Cai and Yeung [7] considers a positive impact of data security. Gkantsidis and Rodriguez [8] found that network coding is quite weak in front of pollution attacks. To prevent this attack, researchers proposed various protocols, e.g., employing a hash function to protect the integrity of a codeword [8]. A different hash function based protocol was also proposed [11]. There is also protocol based on digital signatures from bilinear map [14]. More recent work focuses on constructing protocols which are secure in the standard model, i.e., without assuming the cryptographic hash function is a truly random function [21][17]. There is also another line of work that employs the idea of network coding to construct reliable and distributed storage system [22][23][24], which are orthogonal to our work here. These work focus on how to construct a distributed system using network coding techniques for fast repairing damaged data with multiple clouds; while our work here focus on how to detect whether the outsourced data on a single cloud is modified using the technique that is applied for checking whether a network code is polluted. Le and Markopoulou also considered checking the integrity of network coding enabled cloud storage system [25]. In their work, network coding is employed to repair damaged data fast, but not to help audit the user data.

IX. Conclusion

In this paper, we have designed a general construction of secure cloud storage protocol based on any secure network coding protocol. However, it is not known if a secure network coding protocol can be constructed from a secure cloud storage protocol. It is an interesting future work to consider under what condition this can be done. We also construct two new secure cloud storage protocols, which do not need to assume that a hash function is a truly random function. It is an interesting future work to improve the efficiency of the detailed protocol. There is still space for improving the current implementation if the protocol is to be adopted in practice. Another interesting work is to construct new efficient secure cloud storage protocols based on the current work and existing protocols in the secure network coding area.

References

[1] A. Juels and B. Kaliski Jr, "Pors: Proofs of retrievability for large files," in *ACM Conference on Computer and Communications Security (SP)*, 2007, pp. 584–597.

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

[3] H. Shacham and B. Waters, "Compact proofs of retrievability," in *International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, 2008, pp. 90–107.

[4] C. Wang, S. S. M. Chow, Q. Wang, K. Ren, and W. Lou, "Privacy-preserving public auditing for secure cloud storage," *IEEE Transactions on Computers*, vol. 62, no. 2, pp. 362–375, 2013.

[5] J. Xu and E.-C. Chang, "Towards efficient proofs of retrievability," in *ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, 2012, pp. 79–80.

[6] K. Yang and X. Jia, "An efficient and secure dynamic auditing protocol for data storage in cloud computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 9, pp. 1717–1726, 2013.

[7] N. Cai and R. W. Yeung, "Secure network coding," in *IEEE International Symposium on Information Theory (ISIT)*, 2002, p. 323.

[8] C. Gkantsidis and P. R. Rodriguez, "Cooperative security for network coding file distribution," in *IEEE International Conference on Computer Communications (INFOCOM)*, 2006.

[9] R. Ahlswede, N. Cai, S.-Y. Li, and R. W. Yeung, "Network information flow," *IEEE Transactions on Information Theory*, vol. 46, no. 4, pp. 1204–1216, 2000.

[10] S.-Y. Li, R. W. Yeung, and N. Cai, "Linear network coding," *IEEE Transactions on Information Theory*, vol. 49, no. 2, pp. 371–381, 2003.

[11] Q. Li, J. C. Lui, and D.-M. Chiu, "On the security and efficiency of content distribution via network coding," *IEEE Transactions on Dependable and Secure Computing*, vol. 9, no. 2, pp. 211–221, 2012.

[12] S. Agrawal and D. Boneh, "Homomorphic macs: Mac-based integrity for network coding," in *International Conference on Applied Cryptography and Network Security (ACNS)*, 2009, pp. 292–305.

[13] F. Zhao, T. Kalker, M. Médard, and K. J. Han, "Signatures for content distribution with network coding," in *IEEE International Symposium on Information Theory (ISIT)*, 2007, pp. 556–560.

[14] D. Charles, K. Jain, and K. Lauter, "Signatures for network coding," *International Journal of Information and Coding Theory*, vol. 1, no. 1, pp. 3–14, 2009.

[15] M. R. Albrecht, K. G. Paterson, and G. J. Watson, "Plaintext recovery attacks against ssh," in *IEEE Symposium on Security and Privacy (SP)*, 2009, pp. 16–26.

[16] J. P. Degabriele and K. G. Paterson, "On the (in) security of IPsec in MAC-then-encrypt configurations," in *ACM Conference on Computer and Communications Security (CCS)*, 2010, pp. 493–504.

[17] D. Catalano, D. Fiore, and B. Warinschi, "Efficient network coding signatures in the standard model," in *International Conference on Practice and Theory in Public-Key Cryptography (PKC)*, 2012, pp. 680–696.

[18] Wikipedia, "Wikipedia dump service," http://dumps.wikimedia.org/simplewiki/20130608/, 2013.

[19] F. Chen, "Source code for secure cloud storage based secure network coding," https://sites.google.com/site/chenfeiorange/resource, 2013.

[20] M. A. Shah, R. Swaminathan, and M. Baker, "Privacy-preserving audit and extraction of digital contents," *IACR Cryptology ePrint Archive*, vol. 2008, p. 186, 2008.

[21] R. Gennaro, J. Katz, H. Krawczyk, and T. Rabin, "Secure network coding over the integers," in *International Conference on Practice and Theory in Public-Key Cryptography (PKC)*, 2010, pp. 142–160.

[22] A. G. Dimakis, K. Ramchandran, Y. Wu, and C. Suh, "A survey on network codes for distributed storage," *Proceedings of the IEEE*, vol. 99, no. 3, pp. 476–489, 2011.

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

[24] Y. Hu, P. P. Lee, and K. W. Shum, "Analysis and construction of functional regenerating codes with uncoded repair for distributed storage systems," *IEEE International Conference on Computer Communications (INFOCOM)*, 2013.

[25] A. Le and A. Markopoulou, "Nc-audit: Auditing for network coding storage," in *International Symposium on Network Coding (NetCod)*, 2012, pp. 155–160.