Secure Cloud Storage Meets with Secure Network Coding

Fei Chen, Tao Xiang, Yuanyuan Yang, and Sherman S.M. Chow

Abstract—This paper reveals an 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. Although the two areas are quite different in their nature and are studied independently, we show how to construct a secure cloud storage protocol given any secure network coding protocol. This gives rise to a systematic way to construct secure cloud storage protocols. Our construction is secure under a definition which captures the real world usage of the cloud storage. Furthermore, we propose two specific secure cloud storage protocols based on two recent secure network coding protocols. In particular, we obtain the first publicly verifiable secure cloud storage protocol in the standard model. We also enhance the proposed generic construction to support user anonymity and third-party public auditing, which both have received considerable attention recently. Finally, we prototype the newly proposed protocol and evaluate its performance. Experimental results validate the effectiveness of the protocol.

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

1 Introduction

Cloud storage is being widely adopted due to the popularity of cloud computing. However, recent reports [1], [2] indicate that data loss can occur in cloud storage providers (CSPs). Thus, the problem of checking the integrity of the data in cloud storage, which we referred to as secure cloud storage (SCS), has attracted a lot of attention. On the other hand, networking coding, which was proposed to improve the network capacity, also faces the problem of integrity checking. An intermediate router may intentionally pollute codewords, which results in decoding failures at the endpoints. Checking the integrity of codewords is referred to as the secure network coding problem. Different researchers have studied secure cloud storage and secure network coding independently. Solutions for the former problem, e.g., [3], [4], [5], were proposed only recently. In contrast, the latter area has been examined for more than ten years, e.g., [6], [7].

Secure cloud storage. This problem was first proposed by Juels and Kaliski [3] and Ateniese et al. [4]. Two main entities are involved in these protocols: a user and a cloud storage provider. A user outsources the data to the cloud who promises to store the data. The user then confirms the data integrity by interacting with the cloud using a secure cloud storage protocol. The motivation of data integrity checking lies in several factors. First, due to the poor management of the cloud, the user's data could be lost due to system failures (hardware or software). To cover the accident, the cloud may choose to lie to the user. Second, the cloud has a huge financial incentive to discard the data which is rarely accessed by the user. Ignoring some part of the data helps the cloud to reduce its cost. Third, a cloud could also be hacked and the data could be modified. Fourth, a cloud may behave maliciously because of various possible government pressures. Without a secure cloud storage protocol, the occurrence of these incidents may be hidden by the cloud and gone unnoticed.

The main feature 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 however require the user to store the data locally. Some protocols (e.g., [5], [8], [9], [10]) are publicly verifiable, i.e., anyone besides the user can verify the data integrity; 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 2.

Secure network coding. This problem was first proposed by Cai and Yeung [6] and Gkantsidis and Rodriguez [7]. Network coding is a routing paradigm where a router in the network sends out encoded data packets, which are a function of received data packets, instead of the traditional store-and-forward approach. Encoding can increase the network capacity for multicast tasks. Linear coding, in which a router sends out a linear combination of received data packets, is proved to be sufficient to achieve the increased capacity [11], [12]. This is especially useful in cooperative networks. However, this paradigm also incurs severe security concerns. If an encoded packet is modified illegally, this modification can quickly spread to the whole network

- *F. Chen is with the College of Computer Science and Engineering, Shenzhen University, China. E-mail: fchen@szu.edu.cn.*
- *T. Xiang is with the College of Computer Science, Chongqing University, Chongqing 400044, China. E-mail: txiang@cqu.edu.cn.*
- *Y. Yang is with the Department of Electrical & Computer Engineering, Stony Brook University, Stony Brook, NY 11794. E-mail: yuanyuan.yang@stonybrook.edu.*
- *S.S.M. Chow is with the Department of Information Engineering, The Chinese University of Hong Kong, Hong Kong, China. E-mail: sherman@ie.cuhk.edu.hk.*

Manuscript received 25 June 2014; revised 10 June 2015; accepted 5 July 2015. Date of publication 12 July 2015; date of current version 16 May 2016. Recommended for acceptance by B. Veeravalli. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TC.2015.2456027

because a router encodes all received packets, including the polluted ones. This attack is also known as the pollution attack. The pollution of the codewords could result in data loss when data receivers attempt to decode the data. Thus, when security is a critical concern in a network coding enabled network, the data receivers and routers need to check whether a data packet is polluted.

Essentially, the secure network coding problem is also a type of data integrity checking problem. Many solutions are adopted from traditional data integrity techniques such as cryptographic hash functions [13], MACs [14] or digital signatures [15], [16]. The basic idea is that each codeword in the network needs to be authenticated by checking if the codeword has been modified illegally. The challenge is that the packets in the network are linearly combined by routers and the new packets also need to be authenticated. All current solutions for secure network coding rely on some homomorphic property of the underlying cryptographic techniques. Readers may refer to Section 2 for more detailed related work.

Our work. In this paper, for the first time, we reveal a relationship between these two different areas, i.e., secure cloud storage and secure network coding. Our main result is that we can construct a publicly verifiable secure cloud storage protocol given any publicly verifiable secure linear network coding protocol.

The connection immediately implies that many previous protocols for secure network coding can be transformed for securing cloud storage. With our generic construction, we can *automatically* have many secure cloud storage constructions from existing secure network coding protocols. In contrast, secure cloud storage protocols are currently designed in a rather ad hoc way and there are only few successful protocols.

To demonstrate the power of our construction, we propose two enhanced secure cloud storage protocols that can satisfy the needs of different applications. These new protocols derived from our generic construction also sheds insights on private-key secure cloud storage protocols, although this paper mainly focuses on public-key protocols. Notably, we design the first publicly verifiable secure cloud storage protocol which is secure in the standard model, i.e., without modeling a hash function is a random function when arguing for the security of the protocol. Moreover, we extend our generic construction to support advanced functionalities, in particular, user anonymity, and third-party public auditing. These features have received considerable attention recently.

The security of our generic construction is proved under a security definition which modeling practical application scenarios. We also implement and open-source a prototype of the new publicly-verifiable secure cloud storage protocol and further evaluate its performance. The prototype makes a step forward for the protocol to be adopted in practice.

We hope our work can bring the wisdom behind existing solutions of secure cloud storage to possibly contribute to and provide new perspective to the network coding community.

Organizations. The rest of the paper proceeds as follows. We start by discussing related work in Section 2. We then model the problems of secure cloud storage problem and secure network coding in Sections 3 and 4 respectively. The security requirements of both protocols are also discussed. Section 5 presents our generic construction with formal security analysis. Section 6 detailed our new secure cloud storage protocols based on two recent secure network coding protocols. Section 7 shows how to enhance the generic construction to support user anonymity and third-party public auditing. The experimental performance is reported in Section 8. Finally, we conclude by discussing future work.

2 Related Work

Proof of retrievability was proposed Juels and Kaliski [3] to enable a client to verify if the data outsourced to the cloud is undamaged. The basic idea is that the user embeds some special authentication information (i.e., "sentinels" [3]) in the data at irregular positions. Although the authentication information is not related to the data and is generated randomly, the cloud does not know the specific positions of them. The user can carry out the auditing by asking the cloud to send back the data at some random positions, which either contain the special authentication data or the user's normal data. However, one drawback of this approach is that the total number of the "sentinels" is finite; thus, auditing can only be done a finite number of times.

Ateniese et al. [4] proposed the idea of provable data possession which makes use of homomorphic authentication data. Roughly, computation can be done on a group of data blocks, such that a new authenticator can be computed from the same computation on their authentications. The user can then audits the cloud storage by asking the cloud to send back some computation of the randomly chosen data blocks and an authentication of the computed result. If the authentication is correct, the cloud stores the user's data intact.

To make the query and response compact, two protocols based on pseudorandom functions and a pairing-based signature respectively were proposed with formal security proofs [8]. One of them is extended to support privacy-preserving third-party auditing [5]. A similar protocol which also utilizes pairing is later proposed [10]. Based on a special commitment protocol, a protocol which aims to reduce the communication cost is proposed [9]. There is also some interesting work based on number-theoretic hash functions [17]. All the above protocols are designed in an ad hoc way. In contrast, we provide a systematic and generic design approach.

Network coding was proposed by Ahlswede et al. [11] as a technique to increase the throughput of a multicast network, and later studied by many researchers (e.g., [18], [19]). Regarding security, Cai and Yeung [6] considered the positive impact, while Gkantsidis and Rodriguez [7] found that network coding is quite weak in front of pollution attacks. To prevent this attack, various protocols are proposed, e.g., employing a hash function to protect the integrity of a codeword [7], [13]. There is also some protocol based on pairing-based signatures [16]. Recent work focuses on designing protocols which are secure in the standard model [20], [21].

Orthogonal to our work, another line of research employs network coding to construct reliable and distributed storage systems, such as how to use it to restore lost data by employing multiple clouds [22], [23], [24]. Our work

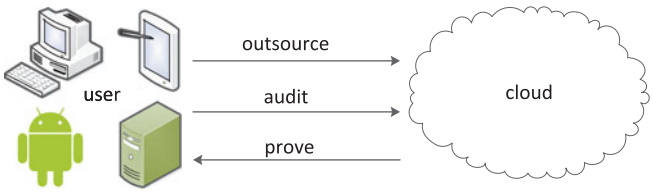


Fig. 1. A secure cloud storage system.

however focuses on how to detect when the outsourced data on a single cloud is modified, via checking if a network code is polluted. Some other works [25], [26], [27] also considered checking the integrity of each data block (i.e., codeword) of network coding enabled cloud storage systems. Yet, traditional data integrity techniques are incorporated to validate the correctness of the data. Network coding does *not* help to audit the user data.

3 Secure Cloud Storage

3.1 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 first *outsources* its data to the cloud. Later, the user periodically performs an *audit* on the integrity of outsourced data. The user can then check whether the proof returned from the cloud is valid or not, meaning that the data remains intact, or obtaining an evidence that the data has been tampered which will possibly incur some further action (which is out of our scope), such as legal action or data recovery.

Similar to previous work [3], [4] and as motivated earlier in this paper, we model the cloud as potentially malicious. We assume the communication between the user and the cloud is authenticated, which can be done by standard techniques. Thus, we can focus our attention on the user and the cloud but not communication.

A secure cloud storage system that enables a user to check the integrity of the outsourced data is expected to be:

- *Correct*. If the cloud indeed stores the whole outsourced data, the cloud can always prove to the user that the data remains intact.
- *Secure*. If the user's data is damaged, the user can detect with high probability in the audit query, even if the cloud tries to cover the event.
- *Efficient*. The computation, storage, and communication cost of both the user and the cloud should be as small as possible.

3.2 High-Level Protocol Specification

Now we abstract a framework for the secure cloud storage problem. Clearly, 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. Denote the secret information by K, the security level by an integer λ, and the user's data by F. The user employs K to process F. The processed data, denoted by F', contains authentication information and is then outsourced to the cloud. On receiving an audit query q from the user, the cloud uses the stored data F' to generate a proof Γ showing that the data is intact. The user then checks whether Γ is valid. Denote the user's verification result by δ. More specifically, a secure cloud storage protocol contains five efficient algorithms $\mathsf{SCS} = (\mathsf{KeyGen}, \mathsf{Outsource}, \mathsf{Audit}, \mathsf{Prove}, \mathsf{Verify})$ as follows:

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

3.3 Understanding Security

This subsection presents a reasonable security definition of the secure cloud storage protocol by abstracting its real-world usage step by step. The key to understand the security is to define the meaning of security exactly.

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 rejects the proof, the user may sue the cloud or follow some other remedy actions; if the user accepts the proof, there are no such actions. Another important issue is how many audit queries and verification results the cloud can get. Our definition allows the malicious cloud to see polynomially many (in security parameter) such queries and verification results; which can cover the user's periodical audit in practice.

Denote a malicious cloud by $\mathcal{A}$, a series of query, proof and verification result by q_i, Γ_i, δ_i where $i = 1, 2, \ldots, \mathrm{poly}(\lambda)$ and $\mathrm{poly}(\lambda)$ is a fixed polynomial in the security parameter λ that bounds the maximal number of such audit/verification results $\mathcal{A}$ 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 defined below:

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

The left hand side denotes the capability of the malicious cloud and the right denotes the cheating behavior. The probability is taken over the random choices of the key generation algorithm and those of 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 intact and the user can retrieve its data. One candidate security intuition is that the probability of a cheating behavior $\Pr[\mathsf{Cheat}]$ is small. However, this is not sufficient and not directly related to whether the data remains intact. It is better if the user can extract its data from the audit queries and 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 computed as:

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

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 because it owns the data. However, we require the total number of queries to be bounded by a polynomial $\mathrm{poly}(\lambda)$ because 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. Thus, 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 for $\mathrm{negl}(\lambda)$ is $\frac{1}{2^\lambda}$. The security intuition 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 $\Pr[\mathsf{Cheat}]$.

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

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

The above definition allows more protocol to be classified as secure. A stronger definition could require $\Pr[\mathsf{Cheat}]$ being negligible and $\Pr[\mathsf{Extract}]$ being as perfect as approximately 1. All protocols in this paper can achieve this stronger definition.

4 Secure Network Coding

4.1 System Model

Fig. 2 shows a typical system that employs the network coding technique. There are three types of entities: *sender*, *router*, and *receiver*. A sender wants to broadcast some data to a group of receivers. The sender divides the data into packets and sends a linear combination of the packets via 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 recover 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 series of packets, the router first checks their correctness, then combines the received correct packets, and finally sends out the combined packet together with the combined authentication information. The combined authentication information is computed according to the details of a specific protocol.

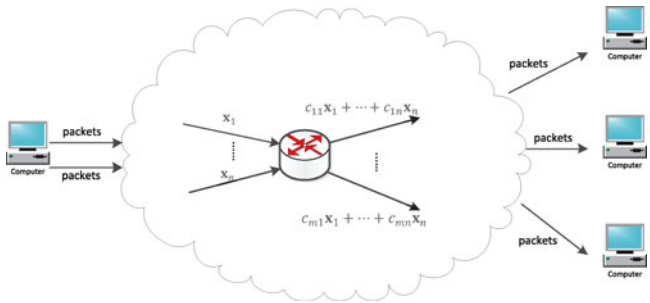


Fig. 2. A secure network coding system.

4.2 Protocol Syntax and Security Definition

We first introduce some notations. Let F be the data to be sent which 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 are linearly combined during the transmission, the coefficients also need to be attached with the packets. Thus, each $\mathbf{v}_i$ is attached with a unit vector $\mathbf{e}_i \in \mathbb{F}_p^m$ where only the ith 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 also updates 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 $\mathsf{SNC} = (\mathsf{KeyGen}, \mathsf{Auth}, \mathsf{Combine}, \mathsf{Verify})$ as follows:

- $\mathsf{KeyGen}(\lambda) \rightarrow (SK, PK)$: On input a security parameter λ, the sender runs this algorithm to generate a secret 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 then 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 $\mathbf{w} \in \mathbb{F}_p^{n+m}$ ant its authentication information t, a receiver or a router runs this algorithm to check whether a packet is modified maliciously. If the packet is correct, it outputs $\delta = 1$, else outputs $\delta = 0$.

Defining security requires understanding the capability of a malicious router, which allows it to observe a lot of packets and their authentication information. A secure network coding protocol should prevent a malicious router from modifying 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, the protocol is insecure. Let $\mathcal{A}$ be a malicious router and $\mathrm{Adv}[\lambda]$ be the probability of finding a forgery packet/authentication pair. $\mathrm{Adv}[\lambda]$ can be defined as:

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

The left hand side shows the information $(\mathbf{w}_j, t_j, \delta_j)\mathcal{A}$ could get in the network and the right hand side denotes the malicious behavior of a router $\mathcal{A}$. 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.*

There are many protocols (e.g., [13], [16], [21]) in the literature that employ the same semantics in the form $\mathsf{SNC} = (\mathsf{KeyGen}, \mathsf{Auth}, \mathsf{Combine}, \mathsf{Verify})$. So we only focus on the high-level specification of a secure network coding protocol in the following discussion.

5 Our Generic Construction

Our goal is to construct a secure cloud storage protocol $\mathsf{SCS} = (\mathsf{KeyGen}, \mathsf{Outsource}, \mathsf{Audit}, \mathsf{Prove}, \mathsf{Verify})$ given a well-designed secure network coding protocol $\mathsf{SNC} = (\mathsf{KeyGen}, \mathsf{Auth}, \mathsf{Combine}, \mathsf{Verify})$.

The basic idea lying behind the generic construction is very intuitive. We can treat the user as a sender who wants to send the data to some receivers; we also treat the user as a data receiver in the network; the cloud is treated as a router in the network. When the user outsources the data, it first divides the data into packets which can be considered as a vector over some finite fields. Then, the user 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 and outputs a linearly encoded packet. The indices of the data packets and the coefficients of the encoding are sent by the user as an audit query. The encoded packet and its authentication information is then sent back to the user as a proof. In this case, the user behaves like a data receiver. By checking whether the returned data packet is valid, the user can judge whether the cloud keeps the outsourced data undamaged.

A naive construction. To implement the above idea, we 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 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 $\mathsf{SNC.Auth}$ and then outsources the processed data $F' = \{(\mathbf{x}_i, t_i)\}_{i=1,\ldots,m}$. 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. For efficiency, we may 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 if the cloud cheats, the user can check the authenticator t of $\mathbf{w}$ using $\mathsf{SNC.Verify}(\mathbf{w}, t)$.

5.1 An Optimized Construction

We observe that the cloud does not need to store the coefficient vectors as in the network coding case. The user knows the coefficients because they are chosen by itself. Thus, considerable storage cost can be saved.

We now summarize our generic secure cloud storage protocol $\mathsf{SCS} = (\mathsf{KeyGen}, \mathsf{Outsource}, \mathsf{Audit}, \mathsf{Prove}, \mathsf{Verify})$ as follows, which is also shown pictorially in Fig. 3.

- $\mathsf{KeyGen}(\lambda) \rightarrow K$: The user determines the finite field $\mathbb{F}_p$ where the network coding works over. The user also determines the packet size n and the total number of packets m. Then the user runs $\mathsf{SNC.KeyGen}$ $(\lambda) \rightarrow (SK, PK)$. The key is $K = (SK, PK)$.
- $\mathsf{Outsource}(F; K) \rightarrow F'$: 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 $\mathsf{SNC.Auth}(\mathbf{x}_i; SK) \rightarrow (\mathbf{x}_i, t_i)$ to get the authentication information. The user then outsources the processed data $F' = \{\mathbf{v}_i, t_i\}_{i=1,\ldots,m}$ together with the public key PK to the cloud. Only the $\mathbf{v}_i$'s are outsourced but not the $\mathbf{x}_i$'s.
- $\mathsf{Audit}(K) \rightarrow q$: The user runs this algorithm to generate a collection of uniformly random numbers $\{i_j, c_j\}_{j=1,\ldots,l}$ where $1 \leq i_j \leq m$ and $c_j \in \mathbb{F}_p$. The user sends the query $q = \{i_j, c_j\}_{j=1,\ldots,l}$ to the cloud. To achieve a good security level, the user sends multiple independent audit queries during one audit process.
- $\mathsf{Prove}(q, F') \rightarrow \Gamma$: 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 $\mathbf{x}_{i_j}$ for all j. The cloud runs $\mathsf{SNC.Combine}(\{\mathbf{u}_i, t_i\}_{i=1,\ldots,l}, \{c_1, \ldots, c_l\}) \rightarrow (\mathbf{w}, t)$ where $\mathbf{u}_i = \mathbf{x}_{i_j}$. 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. The equation $\mathbf{y} = \sum_{j=1}^{l} c_j \cdot \mathbf{v}_{i_j}$ holds.
- $\mathsf{Verify}(q, \Gamma; K) \rightarrow \delta$: 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 $\mathsf{SNC.Verify}(\mathbf{w}, t) \rightarrow \delta$ to get an answer δ. If $\delta = 1$, the

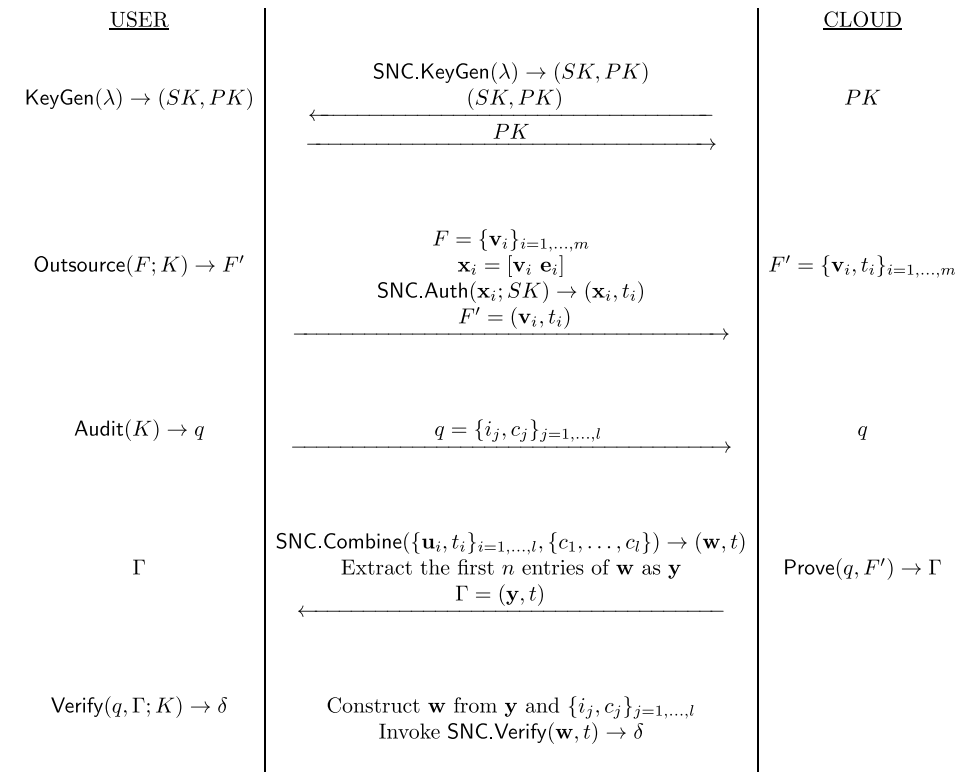


Fig. 3. A generic secure cloud storage protocol based on any secure network coding protocol.

outsourced data remains intact and output 1; else, the outsourced data is damaged and output 0.

5.2 Security Analysis

We show our generic construction is secure under Definition 1 if the protocol is secure with respect to Definition 2. We first present a lemma.

Proposition 3. *Let $r > 0$ be an (arbitrary but fixed) integer, $a_1, \ldots, a_r \in \mathbb{F}_p$ and $a_1 \neq 0$. $\Pr_{x_1,\ldots,x_r}[\sum_{i=1}^{r} a_i \cdot x_i = 0] = \frac{1}{p}$ where x_i is uniformly and independently chosen from $\mathbb{F}_p$.*

Proof. We prove by using conditional probability. Fix the value of $x_2, \ldots, x_r$. Let $c_2, \ldots, c_r \in \mathbb{F}_p$ be constants. Then,

$$\Pr_{x_1}\left[\sum_{i=1}^{r} a_i \cdot x_i = 0 \,|\, x_2 = c_2, \ldots, x_r = c_r\right] = \Pr_{x_1}\left[x_1 = \frac{1}{a_1}\left(-\sum_{i=2}^{r} a_i \cdot c_i\right) \,|\, x_2 = c_2, \ldots, x_r = c_r\right] = \frac{1}{p}.$$

Now it holds that

$$\begin{aligned}
&\Pr_{x_1,\ldots,x_r}\left[\sum_{i=1}^{r} a_i \cdot x_i = 0\right] \\
&= \sum_{c_2,\ldots,c_r \in \mathbb{F}_p} \left[\begin{array}{l}\Pr_{x_2,\ldots,x_r}[x_2 = c_2, \ldots, x_r = c_r] \times \\ \Pr_{x_1,\ldots,x_r}[\sum_{i=1}^{r} a_i \cdot x_i = 0 \,|\, x_2 = c_2, \ldots, x_r = c_r]\end{array}\right] \\
&= \frac{1}{p} \times \sum_{c_2,\ldots,c_r \in \mathbb{F}_p} \Pr_{x_2,\ldots,x_r}[x_2 = c_2, \ldots, x_r = c_r] \\
&= \frac{1}{p} \times 1 = \frac{1}{p}.
\end{aligned}$$

□

The detailed security analysis of the generic construction is accomplished in two parts. In Part I, we show a malicious cloud cannot cheat with a high probability, i.e., $\Pr[\mathsf{Cheat}]$ as defined in Eq. (1) is small. Part I is fairly easy. Part II shows the original data can be extracted by the user with a high probability, and further $\Pr[\mathsf{Extract}] \geq \Pr[\mathsf{Cheat}] - \mathrm{negl}(\lambda)$. Part II deals with some combinatorial and probabilistic argument on the rank of certain random matrix. We will use Proposition 3.

Theorem 4. *The generic construction of* SCS *is secure if the underlying construction component* SNC *is secure.*

Proof. *Part I.* $\Pr[\mathsf{CHEAT}]$ in Eq. (1) is negligible, which follows directly from the security of SNC. Otherwise, we can easily transform the malicious cloud to break the security of the secure network coding protocol.

Part II. We construct an algorithm that a user can employ to extract the outsourced data. Denote the user's data by $F = \{\mathbf{v}_i\}_{i=1,\ldots,m}$ where $\mathbf{v}_i \in \mathbb{F}_p^n$. Algorithm 1 describes the detailed data extraction process.

The correctness of Algorithm $\mathsf{Extract}(\cdot)$ is straightforward. Now we show this algorithm is efficient (runs in polynomial time). Suppose $\mathsf{Extract}(\cdot)$ runs k times before halting. The task is to bound k. $(\mathbf{y}, t)$ is correct with probability $1 - \Pr[\mathsf{CHEAT}]$ and $\Pr[\mathsf{CHEAT}]$ is negligible. So, with probability $1 - k\Pr[\mathsf{CHEAT}]$, $\mathbf{C}_1$ is a special matrix of dimension $m \times k$. Besides, all columns of $\mathbf{C}_1$ are random vectors with only l non-zero elements, and $\mathbf{C}_1^T$ is a $k \times m$ matrix. We want to compute $\Pr_{\mathbf{C}_1}[\mathrm{rank}(\mathbf{C}_1^T) = m]$ as in this case $[\mathbf{v}_1, \ldots, \mathbf{v}_m] \cdot \mathbf{C} = \mathbf{Y}$ has an unique solution and the data can be extracted. We have

$$\begin{aligned}\Pr_{\mathbf{C}_1}[rank(\mathbf{C}_1^T)=m] &= 1-\Pr_{\mathbf{C}_1}[rank(\mathbf{C}_1^T)<m]\\ &= 1-\Pr_{\mathbf{C}_1}[\exists \mathbf{x}\neq \mathbf{0} \text{ s.t. } \mathbf{C}_1^T\mathbf{x}=\mathbf{0}]\\ &\geq 1-\sum_{\mathbf{x}\neq\mathbf{0}}\Pr_{\mathbf{C}_1}[\mathbf{C}_1^T\mathbf{x}=\mathbf{0}].\end{aligned}$$

Now we bound $\Pr[\mathbf{C}_1^T\mathbf{x}=\mathbf{0}]$ for an arbitrary but fixed $\mathbf{x}\neq\mathbf{0}$. Without loss of generality, we focus on the first entry of the vector $\mathbf{C}_1^T\mathbf{x}$. Let E_1 be the event that the first entry is 0. Then, $\Pr[\mathbf{C}_1^T\mathbf{x}=\mathbf{0}]=(\Pr[E_1])^k$. Let E_2 be the event that the non-zero positions of the first row of $\mathbf{C}_1^T$ have no collision with the non-zero positions of $\mathbf{x}$. Suppose $\mathbf{x}$ has $i\geq 1$ non-zero positions. Then

$$\begin{aligned}\Pr[E_2] &= \frac{\binom{m-i}{l}}{\binom{m}{l}}\\ &= \frac{(m-l)!\cdot(m-i)!}{m!\cdot(m-i-l)!}\\ &\leq \frac{(m-l)!\cdot(m-1)!}{m!\cdot(m-1-l)!}\\ &= \frac{m-l}{m}.\end{aligned}$$

Let E_3 be the event that the non-zero positions of the first row of $\mathbf{C}_1^T$ has some collisions with all the non-zero positions of $\mathbf{x}$. Then, $\Pr[E_3]=1-\Pr[E_2]$. According to Proposition 3, $\Pr[E_1\mid E_3]=\frac{1}{p}$. Also we have $\Pr[E_1\mid E_2]=1$ and $1+x\leq e^x$ for any x. Then,

$$\begin{aligned}\Pr[E_1] &= \Pr[E_2]\Pr[E_1\mid E_2]+\Pr[E_3]\Pr[E_1\mid E_3]\\ &\leq \Pr[E_2]+(1-\Pr[E_2])\frac{1}{p}\\ &= \frac{1}{p}+\left(1-\frac{1}{p}\right)\Pr[E_2]\\ &\leq 1-\left(1-\frac{1}{p}\right)\frac{l}{m}\\ &\leq e^{-\left(1-\frac{1}{p}\right)\frac{l}{m}}\end{aligned}$$

and thus $\Pr_{\mathbf{C}_1}[\mathbf{C}_1^T\mathbf{x}=\mathbf{0}]\leq e^{-(1-\frac{1}{p})\frac{kl}{m}}$.

Therefore, we have

$$\begin{aligned}\Pr_{\mathbf{C}_1}[rank(\mathbf{C}_1^T)=m] &\geq 1-\sum_{\mathbf{x}\neq\mathbf{0}}\Pr[\mathbf{C}_1^T\mathbf{x}=\mathbf{0}]\\ &\geq 1-p^m\cdot e^{-\left(1-\frac{1}{p}\right)\frac{kl}{m}}\\ &= 1-e^{-\left(1-\frac{1}{p}\right)\frac{kl}{m}+m\ln p}.\end{aligned}$$

When $k=O(m^2)$, $\Pr_{\mathbf{C}_1}[rank(\mathbf{C}_1^T)=m]$ is approximately equal to 1, which means that we can recover the user's data successfully. The running time of Extract$(\cdot)$ is $O(m^2)$; it is an efficient algorithm. Thus,

$$\begin{aligned}\Pr[\mathsf{EXTRACT}] &= (1-e^{-O(m)})\cdot(1-k\Pr[\mathsf{CHEAT}])\\ &= 1-\mathrm{negl}(\lambda)\\ &\geq \Pr[\mathsf{CHEAT}].\end{aligned}$$

Combining Parts I and II, the proof is completed. □

We think our technique to extract data and its proof is more intuitive and simpler than the existing one [8].

Algorithm 1. Extract$(\cdot)$ - Restore the Outsourced Data

1: The user initializes matrices $\mathbf{C}$, $\mathbf{C}_1$, and $\mathbf{Y}$ with m rows, and indicator variables $count=1, count'=1$.
2: **Repeat**
3: The user sends a uniformly random audit query $q=\{i_j,c_j\}_{j=1,\ldots,l}$ to the cloud.
4: The cloud answers with a proof $(\mathbf{y},t)$.
5: **If** $(\mathbf{y},t)$ passes user's verification **Then**
6: Let $\mathbf{u}$ be a vector s.t. the i_j-entry is set to c_j and all other entries are set to 0.
7: Set the $count'$-column vector of $\mathbf{C}_1$ to $\mathbf{u}$.
8: **If** $\mathbf{u}$ is linearly independent with all the $count-1$ columns of $\mathbf{C}$ **Then**
9: Set the $count$-column vector of $\mathbf{C}$ to $\mathbf{u}$.
10: Set the $count$-column vector of $\mathbf{Y}$ to $\mathbf{y}$
11: Increment $count$ and $count'$ by 1.
12: **End If**
13: **End If**
14: **Until** The rank of $\mathbf{C}$ reaches m.
15: Solve the linear equation $[\mathbf{v}_1,\ldots,\mathbf{v}_m]\cdot\mathbf{C}=\mathbf{Y}$ to recover $F=\{\mathbf{v}_i\}_{i=1,\ldots,m}$.
16: **Return** F

5.3 Further Discussion

Design choices. 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 generic construction of the secure cloud storage protocol decides to treat the whole data as only one generation. This is because that we want to audit the whole data. If multiple generations are adopted, we need to audit each generation to ensure that the user's data remain 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 reducing 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 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 find the cheating behavior if the data is changed 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 research issues. It is also interesting to investigate if a secure network coding protocol can be constructed in general from any secure cloud storage protocol. Yet, there is a mismatch. For example, we can use a keyed cryptographic hash function to compute 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 adopted to be 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 may not be the case for secure cloud

storage. However, there do exist some secure cloud storage protocols that can be converted to secure network coding protocols [8].

6 Concrete Instantiations

6.1 Our First Instantiation

Agrawal and Boneh [14] proposed a secure network coding protocol SNC reviewed as follows.

- $\mathsf{KeyGen}(\lambda) \rightarrow (SK, PK)$: On input a security parameter λ, the sender runs this algorithm to generate a key K_1 for a pseudorandom generator G with range $\mathbb{F}_p^{n+m}$, and a key K_2 for a pseudorandom function F with range $\mathbb{F}_p$. The private key is (K_1, K_2) and the public key is null. The sender also shares the private key with the receiver.
- $\mathsf{Auth}(\mathbf{x}_i; SK) \rightarrow (\mathbf{x}_i, t_i)$: On input the ith packet $\mathbf{x}_i = [\mathbf{v}_i \ \mathbf{e}_i] \in \mathbb{F}_p^{n+m}$ to be sent out in the network, the sender computes an authentication information as follows. First, compute $\mathbf{s} = G(K_1) \in \mathbb{F}_p^{n+m}$ and $b = F(i; K_2)$. Then the authentication information is $t_i = (\mathbf{x}_i, \mathbf{s}) + b$ and $(\mathbf{x}_i, t_i)$ is sent out.
- $\mathsf{Combine}(\{\mathbf{u}_i, t_i\}_{i=1,\ldots,l}, \{c_1, \ldots, c_l\}) \rightarrow (\mathbf{w}, t)$: 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 $\mathbf{w} = \sum_{j=1}^{l} c_j \mathbf{u}_j \in \mathbb{F}_p^{n+m}$. A router also computes the combined authentication information $t = \sum_{j=1}^{l} c_j t_j$. The router sends out $(\mathbf{w}, t)$.
- $\mathsf{Verify}(\mathbf{w}, t) \rightarrow \delta$: On input a packet $\mathbf{w} \in \mathbb{F}_p^{n+m}$ and its authentication information t, the receiver first computes $\mathbf{s} = G(K_1) \in \mathbb{F}_p^{n+m}$ and $b = \sum_{j=1}^{m} \mathbf{w}_{n+j} \cdot F(j; K_2)$. If $(\mathbf{w}, \mathbf{s}) + b = t$, the packet is a correct one and it outputs $\delta = 1$, else outputs $\delta = 0$.

Now we transform this into a secure cloud storage protocol according to our generic construction:

- $\mathsf{KeyGen}(\lambda) \rightarrow K$: Similar to SNC.KeyGen, the user generates K_1 for a pseudorandom generator G and a key K_2 for a pseudorandom function F. The secret key is (K_1, K_2).
- $\mathsf{Outsource}(F; K) \rightarrow F'$: 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 $\mathbb{F}_p^n$. For the ith data block $\mathbf{v}_i$, compute its authentication information as follows. First, compute $\mathbf{x}_i = [\mathbf{v}_i \ \mathbf{e}_i] \in \mathbb{F}_p^{n+m}$ where the ith entry of $\mathbf{e}_i$ is 1 and the other entries are all zero, $\mathbf{s} = G(K_1) \in \mathbb{F}_p^{n+m}$ and $b = F(i; K_2)$. Then the user generates the authentication information as $t_i = (\mathbf{x}_i, \mathbf{s}) + b$. Finally, the user outsources the processed data $F' = \{\mathbf{v}_i, t_i\}_{i=1,\ldots,m}$ to the cloud.
- $\mathsf{Audit}(K) \rightarrow q$: 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 $c_j \in \mathbb{F}_p$. The user sends the query $q = \{i_j, c_j\}_{j=1,\ldots,l}$ to the cloud.
- $\mathsf{Prove}(q, F') \rightarrow \Gamma$: 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 t_{i_j} for each queried data blocks. The cloud then computes $\mathbf{v} = \sum_{j=1}^{l} c_j \mathbf{v}_{i_j}$ and $t = \sum_{j=1}^{l} c_j s_{i_j}$. The cloud sends back $\Gamma = (\mathbf{v}, t)$ as a proof of the corresponding query.
- $\mathsf{Verify}(q, \Gamma; K) \rightarrow \delta$: On input an audit query $q = \{i_j, c_j\}_{j=1,\ldots,l}$, the cloud's proof $\Gamma = (\mathbf{v}, t)$, 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 $\mathbf{v}$, the $(n + i_j)$-entry is c_j, and all other entries are 0. The user first computes $\mathbf{s} = G(K_1) \in \mathbb{F}_p^{n+m}$ and $b = \sum_{j=1}^{m} \mathbf{w}_{n+j} \cdot F(i; K_2)$ and then checks whether $(\mathbf{w}, \mathbf{s}) + b = t$. If they are equal, output $\delta = 1$; else, the data is damaged and output $\delta = 0$.

6.2 Instantiation in the Standard Model

Now we provide another instantiation based on a secure network coding protocol of Catalano et al. [21]. Compared with previous protocols, this protocol is the *first* publicly verifiable secure cloud storage protocol which is secure in the standard model. Since this protocol is more prominent, we also evaluate its theoretical/experimental performance and compare it with some recent secure cloud storage protocols, to be detailed in Section 8.

6.2.1 Protocol Detail

We specify the new protocol by filling all the semantics of a secure cloud storage protocol $\mathsf{SCS} = (\mathsf{KeyGen}, \mathsf{Outsource}, \mathsf{Audit}, \mathsf{Prove}, \mathsf{Verify})$ as follows:

- $\mathsf{KeyGen}(\lambda) \rightarrow K$: The user generates two random primes p, q and sets $N = pq$. The user then generates a prime e with length 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 with N. The secret key is $SK = (p, q)$ and the public key is $PK = (N, e, g, g_1, \ldots, g_n, h_1, \ldots, h_m)$. The key pair is denoted by $K = (SK, PK)$.
- $\mathsf{Outsource}(F; K) \rightarrow F'$: 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.
- $\mathsf{Audit}(K) \rightarrow q$: 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.
- $\mathsf{Prove}(q, F') \rightarrow \Gamma$: 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 block. 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

TABLE 1
Performance Comparison of the New Protocol with Recent Protocols: 'ROM' Denotes the Random Oracle Model; 'crypto' Denotes Cryptographic Operations

Protocols	User			Cloud			Security models
	Computation	Storage	Communication	Computation	Storage	Communication	
This Work	$O(m \cdot n \cdot \text{crypto})$	$O(\lambda)$	$O(l)$	$O(l \cdot n \cdot \text{crypto})$	$O(\lvert F\rvert + \frac{\lvert F\rvert}{m})$	$O(n+\lambda)$	Standard
Yang and Jia [10]	$O(m \cdot n \cdot \text{crypto})$	$O(\lambda)$	$O(l)$	$O(l \cdot n \cdot \text{crypto})$	$O(\lvert F\rvert + \frac{\lvert F\rvert}{m})$	$O(n+\lambda)$	ROM
Xu and Chang [9]	$O(m \cdot n \cdot \text{crypto})$	$O(\lambda)$	$O(l)$	$O(l \cdot n \cdot \text{crypto} + \lambda)$	$O(\lvert F\rvert + \frac{\lvert F\rvert}{m})$	$O(1+\lambda)$	Standard
Wang et al. [5]	$O(\lvert F\rvert \cdot \text{crypto})$	$O(\lambda)$	$O(l)$	$O(l \cdot n \cdot \text{crypto})$	$O(2 \cdot \lvert F\rvert)$	$O(1+\lambda)$	ROM

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 for the corresponding query.

- Verify$(q, \Gamma; K) \rightarrow \delta$: 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 equal, the outsourced data remains intact and output $\delta = 1$; else, the data is damaged and output $\delta = 0$.

6.2.2 Data Dynamics

For support dynamic data, two approaches can be employed: one is based on the hash tree authentication [5]; the other is based on the hash index table [28]. Since these techniques are standard, we do not elaborate it here. Instead, we give an example here for choosing parameters of the new protocol to better understand it. In practice, we can set $e = 257$, $n = 1,024$ and p, q as 1,024-bit primes. Then, a data block is composed of 1,024 sub-blocks, each with 8-bit data; thus the data block is 1 KB. Then, the total number of blocks is $m = \frac{|F|}{8n}$.

6.2.3 Analysis

Security of this protocol follows from Theorem 4 and the security of the underlying secure network coding protocol [21]. The performance of the new protocol is summarized in Table 1. We also compare the performance of the new protocol with some recent secure cloud storage protocols [5], [9], [10]. We focus on asymptotic performance of these 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 construction details, we omit many technical details and only focus on the key influential factors.

Compared with the protocol of Yang and Jia [10], the main difference lies in the security foundation. Our protocol is secure in the standard model while theirs is secure in the random oracle model. The protocol proposed by Xu and Chang [9] is better in the communication cost; but the computation cost for the cloud is higher. The protocol proposed by Wang et al. [5] performs better in the communication cost; but it takes more storage cost, and is only secure in the random oracle model. To conclude, these protocols have different strengths and weaknesses, and they fit with different applications.

7 Extensions

Now we enhance our generic construction to support additional features.

7.1 Anonymity via Security Mediator

7.1.1 Overview

It is more realistic to consider a multi users environment, e.g., in an enterprise setting. However, which user is accessing the cloud may have implication. Recently, Wang et al. proposed to achieve user anonymity through a security mediator [29]. The novelty of such a protocol is that the identity of a user can be hidden from the cloud, thus preserving user anonymity.

The system setup is shown in Fig. 4. A group of users in an organization outsource their data to the cloud through a gateway called security-mediator. The security-mediator serves as a separate entity that can help generating authentication information of the data for the users. However, the users desire to keep their data secret from the security-mediator. When one user wants to outsource some data to the cloud, the user first blinds the data in some secret way, and then sends the blinded data to the security-mediator. On receiving the blinded data, the security-mediator authenticates the blinded data and sends the authentication

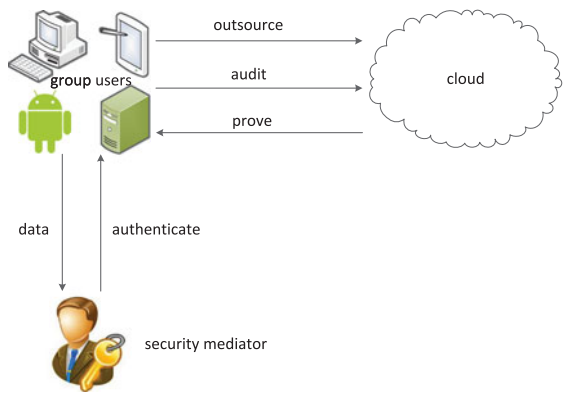


Fig. 4. Security-mediated anonymous cloud storage.

back, from which the data user recovers the real authentication information. Since the authentication information of the outsourced data for every user is generated with the help of the same security-mediator, the cloud cannot differentiate between different users from the outsourced data. In this way, the user anonymity is preserved.

Later, the user sends the data and its authentication information to the cloud via an anonymous channel. In case authentication is needed, this can be done by existing mechanism (e.g., [30]). To ensure the integrity of the outsourced data, one user sends audit queries to the cloud. On receiving an audit query, the cloud answers the query according to the protocol with a proof. Finally, the data user verifies the result returned from the cloud.

We can extend our generic construction using the idea of Wang et al. [29] into a security-mediated secure cloud storage protocol (SM-SCS). Again, we note that the only existing proposal only has a security proof in the random oracle model [29]. We thus obtain an SM-SCS in the standard model if the underlying secure network coding protocol does not need it. It is worth noting that the our construction shows a generic way to design anonymous secure cloud storage protocols for the first time.

7.1.2 Details of Construction

Syntactically, a security-mediated secure cloud storage protocol (SM-SCS) contains seven efficient algorithms (KeyGen, Blind, Unblind, Outsource, Audit, Prove, Verify).

Now we show how to construct a generic security-mediated secure cloud storage protocol SM-SCS = (KeyGen, Blind, Unblind, Outsource, Audit, Prove, Verify) given any secure network coding protocol SNC = (KeyGen, Auth, Combine, Verify). The key is to blind the data which is later authenticated by the security-mediator via a randomization technique. Details are as follows.

- KeyGen$(\lambda) \rightarrow K$: The security-mediator runs this algorithm. SCS.KeyGen employs SNC.KeyGen to generate the key. The security-mediator keeps the secret key only known by itself and shares the public key with both the users and cloud.
- Blind$(\mathbf{w}) \rightarrow \{\mathbf{w}_i\}_{i=1,\ldots,l}$: To get the authentication information for a data block $\mathbf{w}$ in $\mathbb{F}_p^{n+m}$, the user generates l random vectors $\mathbf{w}_i$ such that $\mathbf{w} = \sum_{i=1}^{l} c_i \mathbf{w}_i$ for some secret coefficients c_i's. The user asks the security-mediator to return the authentication information t_i for $\mathbf{w}_i$ using SNC.Auth.
- Unblind$(\{\mathbf{w}_i, t_i, c_i\}) \rightarrow t$: The authentication information has some linear homomorphism as shown in SCS.Prove which is used by cloud to compute the authentication of some linearly combined data blocks. The user thus can also use this linear homomorphism to compute the authentication information for $\mathbf{w}$ in the same way as SCS.Prove. Thus, the authentication information for the unmasked data block $\mathbf{w} = \sum_{i=1}^{l} c_i \mathbf{w}_i$ can be obtained.
- Outsource$(F; K) \rightarrow F'$: The user first blinds the data to be outsourced as in the SM-SCS.Blind algorithm and then obtains the real authentication of the data as in SM-SCS.Unblind. Later, the user sends the data and its authentication information to the cloud.

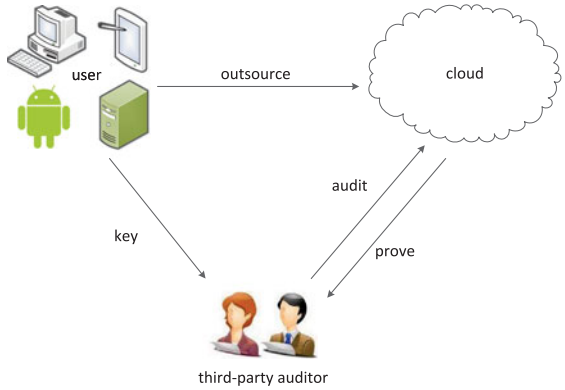


Fig. 5. Third-party auditable secure cloud storage.

- Audit$(K) \rightarrow q$: The user runs this algorithm and the detailed process is the same as SCS.Audit.
- Prove$(q, F') \rightarrow \Gamma$: The cloud runs this algorithm and the detailed process is the same as SCS.Prove.
- Verify$(q, \Gamma; K) \rightarrow \delta$: The user runs this algorithm as same as SCS.Verify, i.e. the user only needs to check if the authentication of the returned result is correct.

7.2 Third-Party Public Auditing

7.2.1 Overview

For a complete outsourcing solution, it is desirable to have the auditing also outsourced to a third-party. This enhancement can reduce the burden of the user by shifting the auditing responsibility to the third-party public auditor. This paradigm is reasonable since the third-party auditor could have more experience and knowledge than the user, and thus the auditing is more convincing and neutral to both parties.

However, the fact that the auditing does not require the original data does not mean that the auditing does not leak the original data. This problem is identified by Wang et al. [5]. Their solution is a zero-knowledge secure cloud storage protocol.

The setup is shown in Fig. 5. The data user generates the key and then outsources the data to the cloud. Any third-party auditor can audit the data by only using the public key. To check if the outsourced data remains intact in the cloud, the auditor sends a series of audit queries to the cloud, and the cloud responds to the queries by returning the proofs. The auditor then can check the proof to see if the data is still intact.

We show how to enhance the generic protocol to support third-party auditing.

7.2.2 Details of Construction

Syntactically, a third-party audit able secure cloud storage protocol (TP-SCS) contains five efficient algorithms TP-SCS = (KeyGen, Outsource, Audit, Prove, Verify) which is as same as the SCS protocol. However, the semantics is improved to incorporate the third-party auditor, as discussed in the system setup. Now the third-party auditor challenges the cloud to check whether the data remains intact instead of the user. To design such an enhanced protocol, the challenge is that the third-party auditor should not understand the user's data due to privacy concerns, but at the same time the third-party auditor can

TABLE 2
Wikipedia Data Set

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

TABLE 3
Additional Storage and Communication Cost: n Denotes the Block Size and m Denotes the Total Number of Blocks

Benchmark	n	m	Storage	Communication
#1	1 KB	3	0.8 KB	376 B
#2	1 KB	1,488	0.54 MB	376 B
#3	1 KB	24,089	8.68 MB	376 B
#4	1 MB	122	0.04 MB	376 B
#5	1 MB	433	0.16 MB	376 B

indeed verify the integrity of the user's data which is outsourced to the cloud.

We now show how to enhance our generic protocol to support third-party auditing. Only the algorithms SCS.Outsource and SCS.Prove in our original protocol $\mathsf{SCS} = (\mathsf{KeyGen}, \mathsf{Outsource}, \mathsf{Audit}, \mathsf{Prove}, \mathsf{Verify})$ need to be enhanced. The key idea hiding the user's data from the third-party auditor is again to blind the data. Given any secure network coding protocol $\mathsf{SNC} = (\mathsf{KeyGen}, \mathsf{Auth}, \mathsf{Combine}, \mathsf{Verify})$, we propose a third-party publicly auditable secure cloud storage protocol $\mathsf{TP\text{-}SCS} = (\mathsf{KeyGen}, \mathsf{Outsource}, \mathsf{Audit}, \mathsf{Prove}, \mathsf{Verify})$ as follows:

- $\mathsf{KeyGen}(\lambda) \rightarrow K$: Everything is the same as SCS.KeyGen as in Section 5. Besides, the user shares the public key with both the third-party auditor and the cloud.
- $\mathsf{Outsource}(F; K) \rightarrow F'$: The user first runs SCS.Outsource as in Section 5. Later, the user also sends some random data blocks $\mathbf{w}_i$ in $\mathbb{F}_p^{n+m}$ with coefficients set to 0 together with their authentications using SNC.Auth.
- $\mathsf{Audit}(K) \rightarrow q$: The third-party auditor runs this algorithm to generate a collection of uniformly random numbers $\{i_j, c_j\}_{j=1,\ldots,l}$ where $1 \leq i_j \leq m$ and $c_j \in \mathbb{F}_p$. The third-party auditor sends the query $q = \{i_j, c_j\}_{j=1,\ldots,l}$ to the cloud. To achieve a good security level, the third-party auditor sends multiple independent audit queries during one auditing process. This process is the same as SCS.Audit.
- $\mathsf{Prove}(q, F') \rightarrow \Gamma$: When receiving a challenge query from the third-party auditor, the cloud computes the result and proof Γ in the same way as in SCS.Prove 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. Since 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 can mask the data and thus the third-party auditor cannot get non-trivial knowledge about the user's data from cloud's response. The returned proof still holds in this case and thus the third-party auditor can indeed check the integrity of the user's data.
- $\mathsf{Verify}(q, \Gamma; K) \rightarrow \delta$: The third-party auditor runs this algorithm as same as SCS.Verify, i.e. the third-party auditor only needs to check whether the authentication of the returned result is correct.

With the enhanced Outsource and Prove algorithms, we have constructed the first *generic* third-party public auditing protocol.

8 Performance Evaluation

8.1 Evaluation Methodology

The setup of the evaluation experiments are as follows. We built a prototype of our newly constructed, the *first* publicly verifiable secure cloud storage protocol in Section 6.2 using Java 7.0. The prototype simulates a cloud and a client. For performance indicators, we consider storage cost, communication cost, and computation cost for both the user and the cloud. The experiments are done on a PC with an Intel i3 3.1G CPU and 4 GB memory. We run the protocol multiple times and average the performance indicators to obtain stable results.

To make all experimental results reproducible, we use the snapshots of the Wikipedia database as our public test data sets [31]. These data are generated by the Wikimedia dump service and could be downloaded online. Besides, we open-source our prototype [32].

8.2 Experimental Results

The data set information is shown in Table 2. To authenticate the outsourced data, we choose p and q to be a 1,024-bit large prime 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 for the authentication information. The experimental result is shown in Table 3. We can see that the storage cost linearly depends on the total number of blocks m. Thus, it is reasonable that we should increase the block size n to increase the 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 percent compared with the original data size. However, block size n cannot be too large in practice since this can result in high communication cost, which depends mainly on the block size. Thus, we need to have a trade-off between storage and communication.

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 the block size; the latter depends on the size of authentication information. The experimental result is also shown in Table 3, which

TABLE 4
Computation Cost: The Time Cost Is Measured in Milliseconds

Benchmark	n	m	Outsource	Audit	Prove	Verify
#1	1 KB	3	832	0.03	828	726
#2	1 KB	1,488	1,119,599	0.04	927	732
#3	1 KB	24,089	18,409,614	2.00	1,009	768
#4	1 MB	122	87,137,897	0.38	970,688	738,411
#5	1 MB	433	325,039,357	4.20	976,570	741,242

only focuses on the additional authentication cost since the block size is fixed. From Table 3, the additional communication cost is small, which is due to the short size of the aggregated authentication information. The size of the authentication information equals the length of 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, namely, the time for outsourcing, auditing, proving, and verifying the data. The experimental result is shown in Table 4. Outsourcing the data takes much longer time than other operations. The time cost grows higher when the data size becomes larger. In theory, it depends on the data size, the block size, and the total number of blocks. The maximal time is 90.3 hours for Data Set 5 of 432 MB. This is very slow; however, this cost is one-time 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 the block size, the length of the audit query, and the 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 the block size and the total number of blocks.

The experiments have shown that the protocol is usable in practice; but we remark that the efficiency could be further optimized.

9 Conclusion

We reveal a relationship between secure cloud storage and secure network coding for the first time. Based on the relationship, we propose a systematic way to construct a generic secure cloud storage protocol based on any secure network coding protocol. As a result, we obtain the first publicly verifiable secure cloud storage protocol which is secure without using the random oracle heuristic. Further, we enhance our generic construction to support user anonymity and third-party public auditing. We hope our open-sourced prototype can make a step towards practical use of secure cloud storage protocols. For future work, it is interesting to design new and efficient secure cloud storage protocols based on our generic construction and existing/future researches on secure network coding protocols. It is also interesting to study the reverse direction, i.e., under what conditions a secure network coding protocol can be constructed from a secure cloud storage protocol. This possibly requires the latter to have some additional properties.

Acknowledgments

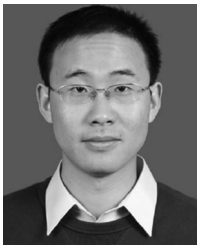
Fei Chen is supported by the Fundamental Research Funds for the Shenzhen University (201533). Tao Xiang is supported by the Natural Science Foundation Project of CQ CSTC (cstc2013jcyjA40001) and the Fundamental Research Funds for the Central Universities (CDJZR13185501). Yuanyuan Yang is supported by the National Natural Science Foundation of China (61373178). Sherman Chow is supported by the Early Career Award and the Early Career Scheme of Research Grants Council, Hong Kong (CUHK 439713), and grants (4055018, 4930034) from The Chinese University of Hong Kong.

References

[1] Y. News. (2013). Cloud computing users are losing data, symantec finds [Online]. Available: http://finance.yahoo.com/news/cloud-computing-users-losing-data-205500612.html
[2] P. Hernande. (2013). Byod, data loss top list of cloud computing challenges [Online]. Available: http://www.datamation.com/cloud-computing/byod-data-loss-top-list-of-cloud-computing-challenges.html
[3] A. Juels and B. Kaliski Jr, "PORs: Proofs of retrievability for large files," in *Proc. ACM Conf. Comput. Commun. Security*, 2007, pp. 584–597.
[4] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable data possession at untrusted stores," in *Proc. ACM Conf. Comput. Commun. Security*, 2007, pp. 598–609.
[5] C. Wang, S. S. M. Chow, Q. Wang, K. Ren, and W. Lou, "Privacy-preserving public auditing for secure cloud storage," *IEEE Trans. Comput.*, vol. 62, no. 2, pp. 362–375, Feb. 2013.
[6] N. Cai and R. W. Yeung, "Secure network coding," in *Proc. IEEE Int. Symp. Inf. Theory*, 2002, p. 323.
[7] C. Gkantsidis and P. R. Rodriguez, "Cooperative security for network coding file distribution," in *Proc. IEEE Int. Conf. Comput. Commun.*, 2006, pp. 1–13.
[8] H. Shacham and B. Waters, "Compact proofs of retrievability," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Security*, 2008, pp. 90–107.
[9] J. Xu and E.-C. Chang, "Towards efficient proofs of retrievability," in *Proc. ACM Symp. Inf., Comput. Commun. Security*, 2012, pp. 79–80.
[10] K. Yang and X. Jia, "An efficient and secure dynamic auditing protocol for data storage in cloud computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 9, pp. 1717–1726, Sep. 2013.
[11] R. Ahlswede, N. Cai, S.-Y. Li, and R. W. Yeung, "Network information flow," *IEEE Trans. Inf. Theory*, vol. 46, no. 4, pp. 1204–1216, Jul. 2000.
[12] S.-Y. Li, R. W. Yeung, and N. Cai, "Linear network coding," *IEEE Trans. Inf. Theory*, vol. 49, no. 2, pp. 371–381, Feb. 2003.
[13] Q. Li, J. C. Lui, and D.-M. Chiu, "On the security and efficiency of content distribution via network coding," *IEEE Trans. Dependable Secure Comput.*, vol. 9, no. 2, pp. 211–221, Mar./Apr. 2012.
[14] S. Agrawal and D. Boneh, "Homomorphic macs: Mac-based integrity for network coding," in *Proc. Int. Conf. Appl. Cryptography Netw. Security*, 2009, pp. 292–305.
[15] F. Zhao, T. Kalker, M. Médard, and K. J. Han, "Signatures for content distribution with network coding," in *Proc. IEEE Int. Symp. Inf. Theory*, 2007, pp. 556–560.
[16] D. Charles, K. Jain, and K. Lauter,"Signatures for network coding," *Int. J. Inf. Coding Theory*, vol. 1, no. 1, pp. 3–14, 2009.
[17] M. A. Shah, R. Swaminathan, and M. Baker, "Privacy-preserving audit and extraction of digital contents," *IACR Cryptol. ePrint Archive*, vol. 2008, p. 186, 2008.
[18] M. Yang and Y. Yang, "A hypergraph approach to linear network coding in multicast networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 21, no. 7, pp. 968–982, Jul. 2010.
[19] M. Yang and Y. Yang, "Applying network coding to peer-to-peer file sharing," *IEEE Trans. Comput.*, vol. 63, no. 8, pp. 1938–1950, Aug. 2014.

[20] R. Gennaro, J. Katz, H. Krawczyk, and T. Rabin, "Secure network coding over the integers," in *Proc. Int. Conf. Practice Theory Public-Key Cryptography*, 2010, pp. 142–160.
[21] D. Catalano, D. Fiore, and B. Warinschi, "Efficient network coding signatures in the standard model," in *Proc. Int. Conf. Practice Theory Public-Key Cryptography*, 2012, pp. 680–696.
[22] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. J. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," *IEEE Trans. Inf. Theory*, vol. 56, no. 9, pp. 4539–4551, Sep. 2010.
[23] A. Bessani, M. Correia, B. Quaresma, F. André, and P. Sousa," Depsky: Dependable and secure storage in a cloud-of-clouds," *ACM Trans. Storage*, vol. 9, no. 4, p. 12, 2013.
[24] H. Chen, Y. Hu, P. Lee, and Y. Tang, "NCCloud: A network-coding-based storage system in a cloud-of-clouds," *IEEE Trans. Comput.*, vol. 63, no. 1, pp. 31–44, Jan. 2014.
[25] B. Chen, R. Curtmola, G. Ateniese, and R. Burns, "Remote data checking for network coding-based distributed storage systems," in *Proc. ACM Workshop Cloud Comput. Security*, 2010, pp. 31–42.
[26] A. Le and A. Markopoulou, "NC-Audit: Auditing for network coding storage," in *Proc. Int. Symp. Netw. Coding*, 2012, pp. 155–160.
[27] H. Chen and P. Lee, "Enabling data integrity protection in regenerating-coding-based cloud storage: Theory and implementation," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 2, pp. 407–416, Feb. 2014.
[28] B. Wang, B. Li, and H. Li, "Public auditing for shared data with efficient user revocation in the cloud," in *Proc. IEEE INFOCOM*, 2013, pp. 2904–2912.
[29] B. Wang, S. S. M. Chow, M. Li, and H. Li, "Storing shared data on the cloud via security-mediator," in *Proc. IEEE 33rd Int. Conf. Distrib. Comput. Syst.*, 2013, pp. 124–133.
[30] S. S. M. Chow, Y. J. He, L. C. K. Hui, and S. Yiu, "SPICE - simple privacy-preserving identity-management for cloud environment," in *Proc. Int. Conf. Appl. Cryptography Netw. Security*, 2012, pp. 526–543.
[31] Wikipedia. (2013). Wikipedia dump service [Online]. Available: http://dumps.wikimedia.org/simplewiki/20130608/
[32] F. Chen. (2013). Source code for secure cloud storage based secure network coding [Online]. Available: https://sites.google.com/site/chenfeiorange/secure-cloud-storage-and-se cure-network-coding

Fei Chen received the BEng and MS degrees in computer science and engineering from Chongqing University, China. He received the PhD degree in computer science and engineering from The Chinese University of Hong Kong. He joined College of Computer Science and Engineering at Shenzhen University, China, as a lecturer in 2015. He visited the Distributed Systems Group at Vienna University of Technology in 2014; he also interned at the Database Team of the Alibaba Group in 2013, and the State Key Laboratory of Information Security of the Chinese Academy of Science in 2009. His research interests include information and network security, data protection, and privacy.



Tao Xiang received the BEng, MS, and PhD degrees in computer science from Chongqing University, China, in 2003, 2005, and 2008, respectively. He is currently a professor in the College of Computer Science, Chongqing University. His research interests include cloud security, wireless security, multimedia security, and cryptography. He has published more than 40 papers on international journals and conferences. He also served as a referee for numerous international journals.

Yuanyuan Yang received the BEng and MS degrees in computer science and engineering from Tsinghua University, Beijing, China, and the MSE and PhD degrees in computer science from Johns Hopkins University, Baltimore, Maryland. She is a professor of computer engineering and computer science at Stony Brook University, New York, and the director in Communications and Devices Division, New York State Center of Excellence in Wireless and Information Technology (CEWIT). Her research interests include data center networks, cloud computing, wireless networks, optical networks, and high-speed networks. She has published more than 300 papers in major journals and refereed conference proceedings and holds seven US patents in these areas. She has served as an associate editor-in-chief and an associated editor for the *IEEE Transactions on Computers* and an associate editor for the *IEEE Transactions on Parallel and Distributed Systems*. She has also served as a general chair, program chair, or a vice chair for several major conferences and a program committee member for numerous conferences. She is a fellow of the IEEE.

Sherman S.M. Chow received the PhD degree from the Courant Institute of Mathematical Sciences, New York University, New York, NY. After receiving the PhD degree, he joined as a research fellow in the Department of Combinatorics and Optimization, University of Waterloo, Waterloo, ON, Canada. He joined the Department of Information Engineering at the Chinese University of Hong Kong, Hong Kong, as an assistant professor in 2012. He interned at NTT Research and Development, Tokyo, Japan, Microsoft Research, Redmond, WA, and the Fuji Xerox Palo Alto Laboratory, Palo Alto, CA. His main interests are in applied cryptography, and security and privacy of networks and distributed systems. He has been serving on the Program Committees of conferences and Editorial Boards on these topics, including ACM CCS, AsiaCCS, AsiaCrypt, ESORICS, Financial Crypt, IEEE CNS, ICDCS, Infocom, PKC, the *International Journal of Information Security*, and the *Journal of Information Security and Applications*. He is a program co-chair of Security in Cloud Computing '15, ISC '14, and ProvSec '14. He received the Early Career Award 2013/14 from the Research Grants Council, Hong Kong.