

On the Security and Efficiency of Content Distribution Via Network Coding

Qiming Li* John C.S. Lui⁺ Dah-Ming Chiu*

*Crypto. & Security Department, Institute for Infocomm Research, Singapore

⁺Computer Science & Eng. Department, The Chinese University of Hong Kong

*Information Eng. Department, The Chinese University of Hong Kong

Abstract—Content distribution via network coding has received a lot of attention lately. However, direct application of network coding may be insecure. In particular, attackers can inject “bogus” data to corrupt the content distribution process so as to hinder the information dispersal or even deplete the network resource. Therefore, content verification is an important and practical issue when network coding is employed. When random linear network coding is used, it is infeasible for the source of the content to sign all the data, and hence the traditional “hash-and-sign” methods are no longer applicable. Recently, a new on-the-fly verification technique is proposed by Krohn et al. (IEEE S&P ’04), which employs a classical homomorphic hash function. However, this technique is difficult to be applied to network coding because of high computational and communication overhead. We explore this issue further by carefully analyzing different types of overhead, and propose methods to help reducing both the computational and communication cost, and provide provable security at the same time.

Keywords: Content distribution, security, verification, network coding.

I. Introduction

For the past few years, there has been an increasing interest on the application of network coding on file distribution. Various researchers have considered the benefit of using network coding on P2P networks for file distribution and multimedia streaming (such as [1]–[5]), while other researchers have considered using network coding on millions of PCs around the Internet for massive distribution of new OS updates and software patches (e.g., the Avalanche project from Microsoft). What we are interested in is the *security* of content distribution schemes using network coding, and how to achieve the security *efficiently*.

An important issue in practical large content delivery in a fully distributed environment is how to maintain the integrity of the data, in the presence of link failures, transmission errors, software and hardware faults, and even malicious attackers. If malicious attackers are able to modify the data in transmission,

or inject arbitrary *bogus* data into the network, they may be able to greatly slow down the content distribution, or even prevent users from getting correct data entirely. In classical content distribution scenarios, data integrity can be checked using a “hash-and-sign” paradigm, where the source employs a collision resistant hash function h to compute hash values of the original data \mathbf{X} and signs the hash value $h(\mathbf{X})$ using a digital signature scheme S with a signing key k . The signature $S_k(h(\mathbf{X}))$ is then used to verify received data Y . However, as we can see later, such methods are not applicable in practical *network coding* based content distribution schemes.

It is first showed, in the seminal work by Ahlswede et al. [6], that if the nodes in the network can perform coding instead of simply forwarding information, multiple sinks in a multicast session can achieve their maximum network flow simultaneously. This technique is referred to as *network coding*. Since then, the topic has been intensively studied, including both theoretical analysis (such as [7]–[9]) and practical discussions (such as [2], [10]–[12]). More details on the literature can be found in Section II-A.

Some classical theoretical results (such as [9]), although provide important insights, would be difficult to apply in practice since they require the knowledge of the network topology during code construction, and require the link failures to follow certain predefined pattern for the code to be reliable. In practice, however, a content distribution network can be very dynamic in terms of the topology, membership, and failures.

Random linear network coding [13] provides a solution to those problems by allowing each node in the network to make *local* decisions. In their setting, the original \mathbf{X} is divided into n blocks $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$, and each node computes and forwards some random linear combination $\mathbf{p} = \sum_{i=1}^n c_i \mathbf{x}_i$ for each of its downstream nodes, together with the coefficients $\mathbf{c} = \langle c_1, \dots, c_n \rangle$. We call the pair (\mathbf{p}, \mathbf{c}) a *packet*. When sufficient linearly independent packets are received, a node would be able to decode the original \mathbf{X} . It is clear that data integrity is even more important in this setting, since, without verification, a node T could combine a damaged (or maliciously modified) packet into all the packets that T generates, and hence all its downstream nodes would received only corrupted data.

Unfortunately, traditional “hash-and-sign” techniques can-

A preliminary version of this paper appeared in ICNP’06.

Qiming Li is with Cryptography and Security Department, Institute for Infocomm Research, A-Star, Singapore. Email: qiming.li@ieee.org

John C.S. Lui is with Computer Science and Engineering Department, Chinese University of Hong Kong. Email: cslui@cse.cuhk.edu.hk

Dah-Ming Chiu is with Information Engineering Department, Chinese University of Hong Kong. Email: dmchiu@ie.cuhk.edu.hk

not be easily applied with random linear network coding. In classical digital signature schemes, only the sender can produce the correct signature of any random combination of data. Hence, the sender would have to pre-compute and distribute the signatures for all possible linear combinations, which is infeasible.

This problem of detecting malicious modifications at intermediate nodes, especially when it is infeasible for the sender to sign all the data being transmitted, is sometimes referred to as *on-the-fly Byzantine fault detection*. Krohn, Freedman and Mazières [14] considered the problem in the context of large content distribution using rateless erasure codes (or fountain codes), and proposed a technique using homomorphic cryptographic hash functions [15]. Hereafter we will refer to this scheme as the KFM scheme. It is noted by Gkantsidis and Rodriguez [10] that the same technique can be adapted in network coding based content distribution. However, both the computational and communication overhead is high.

A simpler and more efficient verification method called *secure random checksum* was proposed by Gkantsidis et al. [16], which comes at the price of weaker security that depends on the secrecy of user-specific parameters. Although computationally efficient, this scheme poses certain limitations on the distribution scenarios, as we will see in Section II-C.

We make a few observations on the differences between the scenarios using erasure code and the network coding. These differences makes the adoption of the KFM scheme [14] very challenging.

Observation 1. The parameters in the KFM scheme allows binary coefficients. Furthermore, the weight of each coefficient vector is a small constant. However, in random linear network coding, it is not clear if there can be too many zero coefficients. Furthermore, the sizes of the coefficients cannot be too small, since otherwise the network coding cannot achieve the theoretical capacity with high probability, and it would be insecure against attacks.

Observation 2. The security of the KFM scheme relies on the size of a security parameter in the hash function (e.g., it should be at least 1024 bit long). In random linear network coding, the size of the modulus will be the same as the size of the smallest data unit as well as that of the coefficients.

Observation 3. The parameters of the hash functions and the hash values of each data block have to be distributed in advance in both settings. Also, during transmission, the coefficient vector has to be transmitted together with each combined data block. However, the binary constant weight vectors in the KFM scheme can be easily compressed, whereas in network coding this is not the case.

These observations give rise to a few challenging issues that need to be addressed to achieve secure and efficient network coding.

Problem 1: The cryptographic hash function used in [14]

is computationally very expensive, even in their probabilistic batch verification variant. This is made worse when the KFM scheme is adopted for random linear network coding, since the random combination coefficients have to be much larger.

Problem 2: The communication overhead in network coding context can be much more significant and cannot be ignored due to the large sizes of the parameters, hash values and coefficient vectors.

We address the first problem by substituting the underlying homomorphic hash function to a recently proposed alternative called VSH, which is much faster [17]. This hash function has an additional advantage that most of the system parameters are fixed, which do not need to be transmitted over the network. Furthermore, we analyze carefully the required sizes of the system parameters, and study how to choose system parameters to avoid unreasonably large communication overhead.

In Section II, we survey previous work on network coding and error detection techniques (II-A), and give detailed descriptions of the KFM scheme (II-B). We observe certain limitations of the secure random checksum scheme [16] in Section II-C. A homomorphic hash function based on VSH is given in Section III, and a basic verification scheme based on this new hash function is given in Section IV. We analyze how to apply the batch verification in Section V. A more efficient *sparse* variant of the random linear network coding is proposed in Section VI, and communication overhead is carefully analyzed in Section VII. We also verify our results with experiments in Section VIII.

II. Background

A. Related Work

It is a well-known graph-theoretic result that the maximum capacity between a source and a sink connected through a network is the same as the maximum network flow f between them. When the network can be viewed as a directed acyclic graph with unit capacity edges, f is also the min-cut of the graph between the source and the sink. However, when there is a single source and multiple sinks, the maximum network flow f may not be achieved. A seminal work of *network coding* [6] reveals that if the nodes in a network can perform coding on the information they receive, it is possible for multiple sinks to achieve their max-flow bound simultaneously through the same network. This elegant result provides new insights into networking today since it now becomes possible to achieve the theoretical capacity bound if one allows the network nodes on the path to perform coding, instead of just the conventional tasks of routing and forwarding.

Later, Li et al. [7] showed that, although the coding performed by the intermediate nodes does not need to be linear, linear network codes are indeed sufficient to achieve the maximum theoretical capacity in acyclic synchronous networks. In their settings, each node computes some linear combination of the information it receives from its upstream nodes, and passes

the results to its downstream nodes. However, to compute the network code (i.e., the correct linear combinations) that is to be performed by the nodes, the topology of the network has to be known beforehand, and has to be fixed during the process of content distribution. Furthermore, their algorithm is exponential in the number of edges in the network.

Koetter and Médard [8], [18] also considered the problem of linear network coding. They improved and extended the results by Li et al. [7], and considered the problem of link failures. They found that a static linear code is sufficient to handle link failures, if the failure pattern is known beforehand. However, as mentioned by Jaggi et al. [9], the code construction algorithm proposed by Koetter et al. still requires checking a polynomial identity with exponentially many coefficients.

Jaggi et al. [9] proposed the first centralized code construction algorithm that runs in polynomial time in the number of edges, the number of sinks, and the minimum size of the min-cut. They also noted that, although the results of Edmonds [19] shows that network coding does not improve the achievable transmission rate when all nodes except the source are sinks, finding the optimal multicast rate without coding is NP-hard. They also showed that if there are some nodes that are neither the source nor the sinks, then multicast with coding can achieve a rate that is $\Omega(\log |V|)$ times the optimal rate without coding, where $|V|$ is the number of nodes in the network. It is also shown in [9] that their method of code construction can handle link failures, provided that the failure pattern is known a priori.

Random network coding was proposed by Ho et al. [13] as a way to ensure the reliability of the network in a distributed setting where the nodes do not know the network topology, which could change over time. In their setting, each node would perform a random linear network coding, and the probability of successful recovery at the sinks can be tightly bounded. Chou et al. [2] proposed a scheme for content distribution based on random network coding in a practical setting, and showed that it can achieve nearly optimal rate using simulations. Recently, Gkantsidis and Rodriguez [3] proposed another scheme for large scale content distribution based on random network coding. They show by simulation that when applied to P2P overlay networks, using network coding can be 20 to 30 percent better than server side coding and 2 to 3 times better than uncoded forwarding, in terms of download time.

With the recent popularity of P2P networks [20], [21], researchers are beginning to consider the problem of on-the-fly Byzantine fault detection in content distribution using random network coding. Authors in [10] noted that the verification techniques proposed by Krohn, Freedman and Mazières [14] can be employed to protect the integrity of the data without the knowledge of the entire content. The verification techniques were originally developed for content distribution using rateless erasure codes and were based on homomorphic cryptographic hash functions [15].

Another simple and efficient on-the-fly verification scheme was proposed by Gkantsidis et al. [12]. Although it may be suitable for certain application scenarios, there are some limitations when it is put under a generic setting, as we will see in Section II-C.

B. The KFM Scheme

1) *Basic Verification Scheme:* The overall picture of the on-the-fly detection technique presented in [14] (which we refer to as the KFM scheme) is illustrated in Fig. 1.

In this scheme, the content \mathbf{X} is divided into n blocks $\mathbf{x}_1, \dots, \mathbf{x}_n$, and each block \mathbf{x}_i is further divided into m sub-blocks $x_{i,1}, \dots, x_{i,m}$, where each $x_{i,j}$ can be represented by an element in the multiplicative group \mathbb{Z}_p^* for some large prime p .

A hash function \mathcal{H} is then applied on each block to obtain the hash values h_1, \dots, h_n . In particular, the hash function uses m generators $g_1, \dots, g_m \in \mathbb{Z}_p^*$, and the hash value h_i of the i -th block is computed as $h_i = \prod_{j=1}^m g_j^{x_{i,j}} \pmod p$.

Clearly, the hash function \mathcal{H} is homomorphic in the sense that for any two blocks \mathbf{x}_i and \mathbf{x}_j , it holds that $\mathcal{H}(\mathbf{x}_i)\mathcal{H}(\mathbf{x}_j) = \mathcal{H}(\mathbf{x}_i + \mathbf{x}_j)$. These hash values are distributed to all the nodes reliably in advance. It is suggested in [14] that the same technique can be used recursively on the hash values until the final hash value is small enough to be distributed without coding. After receiving a coded block \mathbf{x} , which is a linear combination of the original n blocks with coefficients $C = \langle c_1, \dots, c_n \rangle$, a node will be able to verify the integrity of \mathbf{x} using \mathbf{x} , C , and the hash values h_1, \dots, h_n , making use of the homomorphic property of \mathcal{H} . In particular, the node checks if the following holds

$$\mathcal{H}(\mathbf{x}) = \prod_{i=1}^n h_i^{c_i} \pmod p. \quad (1)$$

2) *Limitations:* There are two inherent *limitations* in the above scheme. First, it is computationally expensive to compute the underlying hash function \mathcal{H} . Secondly, all parameters of the scheme, including all the hash values, must be distributed in advance, even to verify a single data packet.

In [14], the first problem of expensive computation is dealt with by using two techniques. On one hand, the parameters are chosen such that all coefficients in C are very small, and many of them are actually zeros. On the other hand, verification of multiple data packets can be done in batches to reduce the overall computational cost. The second problem, however, is tackled by choosing the parameters carefully such that the overhead due to the hash values is less than 1%, which is acceptable in many practical scenarios. Nevertheless, when the original data is relatively large (say, 1GB), the start-up delay caused by the transmission of the hash values may not be tolerable, especially in dynamic networks where nodes frequently join and leave.

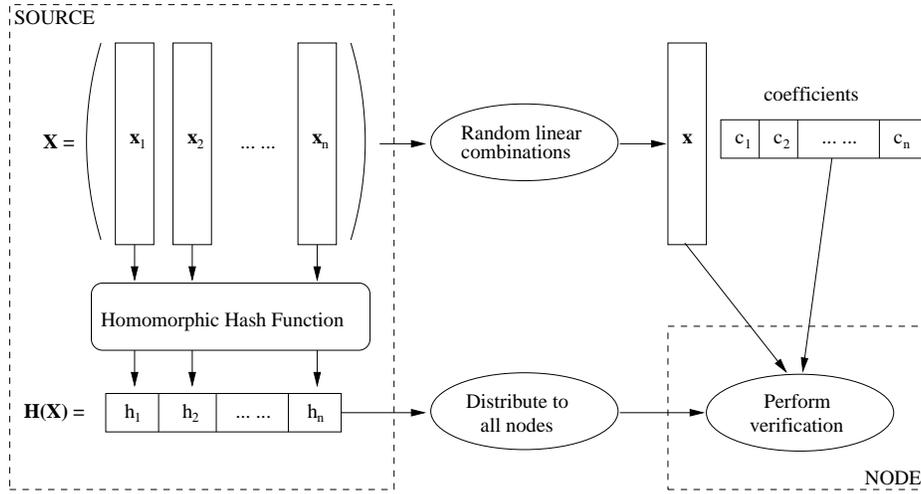


Fig. 1. On-the-fly Byzantine fault detection [14].

3) *Challenges*: It is worth to note that the KFM scheme is intended to be applied on rateless erasure codes, where only the source of the content performs the coding, which is in contrast with the idea of network coding, where all intermediate nodes can participate in the coding. The difference in these settings leads to challenging problems when adopting this scheme in random network coding.

Basically, to allow intermediate nodes freely combine data blocks, the combination coefficients has to be chosen from the same group as the sub-blocks (which is the smallest unit in combination). Hence, for a randomly combined block, the size of the coefficients will be $n|p|$ (where $|\cdot|$ denotes the bit length). Clearly, $|p|$ cannot be small (e.g., at least 1024), since otherwise the hash function can be easily broken. Furthermore, if the size of the content is large, n cannot be too small either, since otherwise m would be too large to make the distribution of parameters g_1, \dots, g_m prohibitively expensive.

As a result, this not only introduces extra communication overhead due to either the hash parameters or the coefficients, but also makes the computation of (1) (hence the verification) much more expensive.

C. The Secure Random Checksum Scheme

An alternative to the expensive cryptographic hash function as used in [14] is proposed in [16], which is called the Secure Random Checksum (SRC). Their main idea is as the following. Before the actual downloading commences, each node retrieves a *checksum* for each block of data from the source via a secret channel. Each checksum is computed as a random combination of all the sub-blocks in a block, and the coefficients are different for each node. These checksums are also homomorphic, so that they can be used to check the integrity of any received packet in a way similar to homomorphic hash functions. Since only linear combinations are involved in the computation of these checksums, the verification can be very efficient.

We note that in this scheme, every node needs to download a set of distinct checksums directly from the source. This creates a centralized downloading scenario with a smaller content size (which is the size of the checksums), which may lead to two problems. First, this poses limits on the scalability of the scheme, since the source could be overwhelmed by the requests to download checksums. Whereas in the case of homomorphic hashes, although the hash values have larger sizes, it is not necessary to download them directly from the source but they can be instead obtained from peers, and no additional secure channel is required. Secondly, the source is required to be online until all the nodes have received checksums from it, which makes it difficult for dynamic networks where nodes are frequently leaving and joining the network. To some extent, the use of such checksums weakens the potential advantages one could expect from a distributed content distribution scheme.

- \mathcal{H} A homomorphic hash function based on VSH.
- \mathbb{Z}_p^* A multiplicative group with large prime modulus p .
- \mathcal{G} A subgroup of \mathbb{Z}_p^* of prime order q . $p = \alpha q + 1$.
- λ Bit length of prime modulus p ($\lambda = |p|$).
- γ Bit length of prime order q ($\gamma = |q|$).
- \mathbf{X} Original content to be distributed.
- n Number of blocks of \mathbf{X} . That is, $\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_n)$.
- m Number of subblocks per block. Each subblock is an element of \mathcal{G} .
- h_i The hash value of the i -th block.
- \mathbf{x} A block of data, or a combination of blocks.
- \mathbf{c} A coefficient vector of length n .
- \mathbf{p} A packet, consisting of a combined block and a coefficient vector.
- b The number of packets in one batch.
- θ Number of packets to combine in the sparse random linear network coding.
- H_1 Hash value computed from a data block ($H_1 = \mathcal{H}(\mathbf{x})$).
- H_2 Hash value computed from hash values h_i 's using the homomorphic property of the hash function.

TABLE I
SYMBOLS USED IN THIS PAPER

III. A Homomorphic Hash Function Based on VSH

The proposed basic scheme is based on the non-trapdoor variant (VSH-DL) of the Very Smooth Hash (VSH) function due to Contini et al. [17]. The rationale behind VSH and its variants is that using smaller primes as group generators would greatly improve the computational efficiency of hash functions that involve many exponentiations. The VSH functions, however, do not have the homomorphic property that would be required by the on-the-fly verification process.

In this paper, we propose a homomorphic hash function based on the same idea of VSH-DL, which we will describe in detail in the next subsection. The homomorphic property is obtained by re-arranging the order of the bits in the input message blocks before applying VSH-DL. For readability, all symbols used are presented in Table I.

A. A Homomorphic VSH-DL

Let p be a large strong prime, such that there is another large prime q divides $p - 1$. That is, there is a positive integer α such that $p = \alpha q + 1$. As a result, there exist a multiplicative subgroup \mathcal{G} in \mathbb{Z}_p^* with order q . Furthermore, for any $x \in \mathbb{Z}_p$, let $y = x^\alpha \pmod p$, if $y \neq 1 \pmod p$, y must be in \mathcal{G} . For convenience, let $|p| = \lambda$ and $|q| = \gamma$, where $|\cdot|$ denotes the bit length.

Let p_1, \dots, p_m be m prime numbers, such that $m < \gamma^c$ for some constant c . In other words, m is bounded by some polynomial in γ . In practice, we can choose those primes to be the m smallest prime numbers, such that $p_i^\alpha \neq 1 \pmod p$. That is, we can choose $p_1 = 2$, $p_2 = 3$, and so on, and skip those primes whose order is not q . When q is much larger than α (e.g., $\alpha = 2$), the probability that a random small prime p_i satisfies the condition $p_i^\alpha \neq 1 \pmod p$ is high.

Assume that a *message* is a vector of the form: $\mathbf{x} = (x_1, \dots, x_m)$ where $x_i \in \mathbb{Z}_q$ for $1 \leq i \leq m$. The hash of \mathbf{x} is computed as

$$\mathcal{H}(\mathbf{x}) = \prod_{i=1}^m p_i^{\alpha x_i} \pmod p. \quad (2)$$

It is worth to note that the original VSH-DL function is equivalently computing the same function with $\alpha = 2$, but explicitly using the parallel exponentiation algorithm due to Bellare et al. [22] and used in [14]¹.

Homomorphic Property: For any two messages $\mathbf{x} = (x_1, \dots, x_m)$ and $\mathbf{y} = (y_1, \dots, y_m)$, it is not difficult to see that

$$\mathcal{H}(\mathbf{x})\mathcal{H}(\mathbf{y}) = \mathcal{H}(\mathbf{x} + \mathbf{y}). \quad (3)$$

In other words, the hash function \mathcal{H} is $(+, \times)$ -homomorphic.

¹There is an off-by-one error in the algorithm due to Bellare et al. [22], which is corrected in [14]

B. Security of the Hash Function

The security of \mathcal{H} is defined in terms of the difficulty in finding collisions. To precisely define the security, we follow the commonly used notions of negligible functions and computational feasibility in cryptography.

Definition 1 A function $f(n)$ is called negligible w.r.t. n if for any positive polynomial $\text{poly}(\cdot)$, for all large enough n , it holds that $f(n) < 1/\text{poly}(n)$.

Furthermore, we are mainly interested in the collision resistance of the hash function.

Definition 2 A hash function \mathcal{H} is collision resistant if for any polynomial-time probabilistic algorithm A , the probability $\Pr[A(p, q) = (\mathbf{x}, \mathbf{y}) \text{ s.t. } \mathcal{H}(\mathbf{x}) = \mathcal{H}(\mathbf{y})]$ is negligible w.r.t. $|q|$.

In other words, \mathcal{H} is collision resistant if it is computationally infeasible to find two messages \mathbf{x} and \mathbf{y} that produce the same hash value.

Next, we define a generalized version of the VSDL problem as in [17].

Definition 3 (GVSDL) Let p be a large prime, and let q be a large prime such that $p = \alpha q + 1$ for some positive integer α . Let p_1, \dots, p_m be the m smallest prime numbers such that $p_i^\alpha \neq 1 \pmod p$ for $1 \leq i \leq m$, where $m \leq \gamma^c$ for some constant c . GVSDL is the following problem: Given the aforementioned parameters, compute $e_1, \dots, e_m \in \mathbb{Z}_q$, such that at least one of the e_i 's is non-zero, and

$$\prod_{i=1}^m p_i^{\alpha e_i} = 1 \pmod p. \quad (4)$$

When $\alpha = 2$, the above definition reduces to the VSDL problem. We say that an algorithm A solves a given GVSDL problem instance (p, q) if $A(p, q) = (e_1, \dots, e_m)$ such that some e_i is non-zero, and (4) holds. Note that p_1, \dots, p_m are implicitly specified once p and q are given, and they can be computed efficiently. Furthermore, we say that GVSDL is *computationally hard* if for any polynomial-time probabilistic algorithm A , the probability that A solves a random GVSDL problem instance (p, q) is negligible w.r.t. $\gamma = |q|$. The probability is taken over the random choices of p, q and the internal coin tosses made by A .

Following the result in [17], we can see that the hash function \mathcal{H} is collision resistant if the GVSDL problem is computationally hard.

Lemma 4 The hash function \mathcal{H} is collision resistant if GVSDL is computationally hard w.r.t. $\gamma = |q|$.

Proof: Suppose on the contrary that there exists an attacker A that can compute a collision of \mathcal{H} with a probability that is not negligible, we show that we can build another

algorithm B that makes use of A to solve the GVSDL problem also with a probability that is not negligible.

In particular, given (p, q) , which is an instance of a GVSDL problem, and an attacker A , we use A to find two messages $\mathbf{x} = (x_1, \dots, x_m)$ and $\mathbf{y} = (y_1, \dots, y_m)$ such that $\mathbf{x} \neq \mathbf{y}$ but $\mathcal{H}(\mathbf{x}) = \mathcal{H}(\mathbf{y})$. In this case, according to (2), we have

$$\prod_{i=1}^m p_i^{\alpha x_i} = \prod_{i=1}^m p_i^{\alpha y_i} \pmod{p}.$$

Let $z_i \triangleq x_i - y_i \pmod{q}$, the above is equivalent to

$$\prod_{i=1}^m p_i^{\alpha z_i} = 1 \pmod{p}.$$

This is exactly a solution to the GVSDL problem instance. Note that at least some z_i 's are non-zero because $\mathbf{x} \neq \mathbf{y}$. If A succeeds with probability p_a , then we can also find the solution z_i 's with probability p_a . ■

IV. The Basic Integrity Verification Scheme

A. The Basic Scheme

Our proposed scheme consists of two algorithms, namely the *encoding* algorithm where the original data are prepared for distribution, and the *verification* algorithm, which is used by individual nodes to verify the integrity of the received data.

1) *Encoding*: Let the parameters p, q, m and p_1, \dots, p_m be chosen as in Section III-A. Given any binary string X , let n be the smallest positive integer such that $|X| < mn(\gamma - 1) - 1$. Assume that $n < \text{poly}(\gamma)$ for some positive polynomial poly . We also assume that X is compressed, such that the bits are random.

In this way, we can always pad the original X properly (e.g., with a one followed by zeros) such that the result can be divided into small pieces of $\gamma - 1$ bits each. In other words, we can always encode the data into the form $\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_n)$, where $\mathbf{x}_i = (x_{i,1}, \dots, x_{i,m})^T$ and each $x_{i,j}$ is of length $\gamma - 1$, and can be considered as an element in \mathbb{Z}_q for all $1 \leq i \leq n$ and $1 \leq j \leq m$.

We will call \mathbf{x}_i as the i -th *block*, and each $x_{i,j}$ as the j -th *sub-block* of the i -th block. Now, given \mathbf{X} , the encoder computes

$$h_i = \mathcal{H}(\mathbf{x}_i) = \prod_{j=1}^m p_j^{\alpha x_{i,j}} \pmod{p} \quad (5)$$

for each $1 \leq i \leq n$.

2) *Basic Verification Algorithm*: During verification, each network node is given a packet (\mathbf{x}, \mathbf{c}) and system parameters. In the case where this packet is not tampered with, $\mathbf{c} = (c_1, \dots, c_n)$ are the coefficients where each $c_i \in \mathbb{Z}_q$, \mathbf{x} is the linear combination $\mathbf{x} = \sum_{i=1}^n c_i \hat{\mathbf{x}}_i \pmod{\mathbb{Z}_q}$.

Each node can verify the integrity of the packet as the following.

- 1) Compute the hash value $H_1 = \mathcal{H}(\mathbf{x})$.
- 2) Compute $H_2 = \prod_{i=1}^n h_i^{c_i} \pmod{p}$.
- 3) Verify that $H_1 = H_2$.

It is worth to note that once an intermediate node has received the parameters p, q and m , it will be able to compute α and p_1, \dots, p_m locally. Therefore, those prime bases do not need to be distributed. The hash values, however, still need to be distributed. Nevertheless, we will see that in any reasonable setting the communication overhead is low.

B. Security Analysis

Intuitively, an attack is considered as successful if the attacker, with the full knowledge of the content being distributed and all the public parameters, can generate a packet (\mathbf{p}, \mathbf{c}) such that \mathbf{p} is not a linear combination of the original data specified by \mathbf{c} but the packet still passes the verification. Hence, we have the following definitions of integrity attackers and the security.

Definition 5 An integrity attacker A is a probabilistic polynomial-time algorithm, such that given parameters p, q, m, p_1, \dots, p_m and the encoded content $\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_n)$, the attacker A finds \mathbf{y} and $\mathbf{c} = (c_1, \dots, c_n)$ so that for $\mathbf{x} \triangleq \sum_{i=1}^n c_i \mathbf{x}_i$, we have $\mathbf{y} \neq \mathbf{x}$ yet $\mathcal{H}(\mathbf{x}) = \mathcal{H}(\mathbf{y})$.

Definition 6 The basic scheme is secure if there does not exist any integrity attacker that can succeed with a probability that is not negligible w.r.t. γ .

We show that the basic scheme is secure, since otherwise we can solve the GVSDL problem.

Theorem 7 The basic scheme is secure if GVSDL is computationally hard w.r.t. γ .

Proof: Essentially, we need to show that if there exists an integrity attacker A that succeeds with probability P that is not negligible in γ , we can construct a probabilistic algorithm B that solves the GVSDL problem with the same probability P , which contradicts with the assumption that GVSDL is hard. In particular, we construct an algorithm B as follows. Given input p, q, m, p_1, \dots, p_m as in Definition 3, the algorithm B does the following steps.

- 1) Randomly select $n \in [2, \text{poly}(\gamma)]$, for some positive polynomial poly .
- 2) Randomly select $x_{i,j} \in \mathbb{Z}_q$ for all $1 \leq i \leq m$ and $1 \leq j \leq n$. Denote $\mathbf{x}_i \triangleq (x_{i,1}, \dots, x_{i,m})$ and $\mathbf{X} \triangleq (\mathbf{x}_1, \dots, \mathbf{x}_n)$.
- 3) Invoke A with p, q, m, p_1, \dots, p_m and \mathbf{X} as inputs. If A fails, halt and declare failure. If A succeeds, let $\mathbf{y} = (y_1, \dots, y_m)$ and $\mathbf{c} = (c_1, \dots, c_n)$ be the output.
- 4) Output e_1, \dots, e_m , where $e_i = y_i - \sum_{j=1}^n x_{i,j} c_j \pmod{q}$ for $1 \leq i \leq m$.

Clearly, algorithm B runs in polynomial time. When A succeeds in Step 3, we know that $\mathbf{y} \neq \sum_{j=1}^n c_j \mathbf{x}_j$, hence at

least one of the e_i 's is non-zero. Therefore, B succeeds if and only if A succeeds. In other words, B succeeds with the same non-negligible probability P , hence creating a contradiction. ■

V. Batch Verification

A. The Baseline Batch Verification Scheme

To reduce the computational cost of the verification, a batch of packets can be verified at the same time. In particular, after a node has received b packets $((\mathbf{y}_1, \mathbf{c}_1), (\mathbf{y}_2, \mathbf{c}_2), \dots, (\mathbf{y}_b, \mathbf{c}_b))$ (not necessarily from the same source), the node can verify all the packets as follows.

- 1) Randomly choose b numbers $r_1, \dots, r_b \in \mathbb{Z}_q$.
- 2) Compute $\mathbf{w} = \sum_{i=1}^b r_i \mathbf{y}_i \pmod q$.
- 3) Compute $\mathbf{v} = \sum_{i=1}^b r_i \mathbf{c}_i \pmod q$.
- 4) Verify the integrity of the packet (\mathbf{w}, \mathbf{v}) using the basic integrity verification scheme.

Due to the homomorphic property of the hash function, we can see that if the verification in Step 4 fails, then at least one of the packets is corrupted. However, if the batch of packets pass the verification, it is still possible that some packets are corrupted but could not be detected by the verification algorithm. Hence we need to analyze the security more carefully.

B. Security Analysis

We first extend the definition of an integrity attacker to work on a batch of packets.

Definition 8 *A successful batch integrity attacker A is a probabilistic polynomial-time algorithm, such that given parameters p, q, m, p_1, \dots, p_m and the encoded content $\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_n)$, the attacker A computes b packets $((\mathbf{y}_1, \mathbf{c}_1), (\mathbf{y}_2, \mathbf{c}_2), \dots, (\mathbf{y}_b, \mathbf{c}_b))$, where at least for some \mathbf{y}_i and $\mathbf{c}_i = (c_{i,1}, \dots, c_{i,n})$, we have $\mathbf{y}_i \neq \sum_{i=1}^n c_i \mathbf{x}_i$, but the packets pass the batch verification algorithm with a probability that is not negligible.*

We can show that, to come up with a batch of packets that pass the verification above is not easier than breaking the basic verification scheme itself.

Corollary 9 *The batch verification scheme is secure (i.e., no successful batch integrity attacker exists) if the basic scheme is secure.*

Proof: Suppose a successful batch integrity attacker A exists. We construct another attack algorithm B as the following. Given parameters $b, p, q, m, p_1, \dots, p_m$ and the encoded content $\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_n)$, the attacker B performs the following steps.

- 1) Invoke A to obtain b packets $((\mathbf{y}_1, \mathbf{c}_1), \dots, (\mathbf{y}_b, \mathbf{c}_b))$. Let $Y \triangleq \{(\mathbf{y}_i, \mathbf{c}_i) \mid \mathbf{y}_i \neq \sum_{j=1}^n c_{i,j} \mathbf{x}_j\}$, where we denote $\mathbf{c}_i = (c_{i,1}, \dots, c_{i,n})$. That is, Y is the set of corrupt packets.
- 2) For every packet $(\mathbf{y}_i, \mathbf{c}_i)$ in Y , check if it passes the basic integrity verification. If a packet $(\mathbf{y}', \mathbf{c}')$ in Y passes the verification, output $(\mathbf{y}', \mathbf{c}')$ and halt. Otherwise do the following.
- 3) Randomly choose b coefficients $r_1, \dots, r_b \in \mathbb{Z}_q$.
- 4) Compute $\mathbf{w} = \sum_{i=1}^b r_i \mathbf{y}_i \pmod q$.
- 5) Compute $\mathbf{v} = \sum_{i=1}^b r_i \mathbf{c}_i \pmod q$.
- 6) Output packet (\mathbf{w}, \mathbf{v}) .

If the algorithm halts at Step 2, clearly it has already successfully attacked the basic verification scheme. Otherwise, by the definition of the attacker A , the packet (\mathbf{w}, \mathbf{v}) passes the basic verification with a probability p_a that is not negligible.

If (\mathbf{w}, \mathbf{v}) is indeed corrupted (i.e., \mathbf{w} is not the linear combination of \mathbf{X} with coefficients in \mathbf{v}), such a packet would be considered as a successful attack on the basic integrity verification scheme, and the attacker B would be successful. Otherwise, when (\mathbf{w}, \mathbf{v}) appears to be not corrupted (e.g., when the coefficients r_i 's chosen for the corrupted packets happen to be 0), the attack would fail. However, since the coefficients r_i 's are randomly chosen, the probability that the second case happens is exponentially small.

Therefore the algorithm B can successfully attack the basic verification scheme with probability p_b that is not negligible (since $p_b - p_a$ is negligible). Hence the corollary follows. ■

When the batch verification technique is used, it is clear that the computational cost can be reduced roughly by a factor of b , since we perform verification on b packets at a time, and the additional cost introduced by the random linear combination of the packets is not significant compared with the cost of the basic verification on the combined packet. Nevertheless, it should be noted that b cannot be too large for typical applications since it will introduce additional delay in the content distribution process, since a node needs to wait until b packets to arrive before the verification can be done.

C. An Enhanced Scheme

The cost of verification in the batch verification can be further reduced. Recall that to verify the a packet (\mathbf{w}, \mathbf{v}) using the basic verification scheme, we must compute H_1 and H_2 as below and check if they are the same.

$$H_1 = \prod_{j=1}^m p_j^{\alpha w_j}, \quad H_2 = \prod_{j=1}^n h_j^{v_j} \quad (6)$$

where $\mathbf{w} = (w_1, \dots, w_m)$, $\mathbf{v} = (v_1, \dots, v_n)$, and h_j is the hash value of the j -th block. Intuitively, if we can make the exponents in the computation as small as possible, the computation cost can be reduced.

To see how we can make w_j 's small, let us consider the batch verification process more carefully in matrix form. Let us denote a batch of packets by (\mathbf{Y}, \mathbf{C}) , where $\mathbf{Y} = (\mathbf{y}_1, \dots, \mathbf{y}_b)$ is the actual data (when not corrupted), and $\mathbf{C} = (\mathbf{c}_1, \dots, \mathbf{c}_b)$ are the coefficient vectors. During the batch verification, another random vector $\mathbf{r} = (r_1, \dots, r_b)^T$ is used, where $\mathbf{w} = \mathbf{Y}\mathbf{r}$ and $\mathbf{v} = \mathbf{C}\mathbf{r}$. Hence, if we instead choose \mathbf{w} (or part of \mathbf{w}) and find the corresponding \mathbf{r} , we may be able to greatly reduce the computation cost. In particular, if $b \geq m$, we can randomly choose \mathbf{w} and find a solution to $\mathbf{w} = \mathbf{Y}\mathbf{r}$ provided that the rank of \mathbf{Y} is m . Even if the rank of \mathbf{Y} is not m or $b < m$, we can still choose the values of some of the components of \mathbf{w} , remove some rows in \mathbf{Y} and solve the reduced system to find \mathbf{r} , and finally compute the rest of the components in \mathbf{w} . If b is not too small compared with m , this method could save a large portion of the computation cost.

A similar method can also be applied on the computation of H_2 to make \mathbf{v} very small. This can be useful when b is not small compared with n . However, clearly it cannot be used simultaneously with the first method. The actual method to apply would be determined by the choice of the parameters m , n and b .

It is worth to note that the effects of reducing \mathbf{w} and \mathbf{v} are not the same. This is because p_j 's are in general much smaller than h_j 's, hence the computation of H_1 is much more efficient than H_2 when $m = n$. As a result, the overall computation cost has to be considered when deciding which function is to be optimized.

VI. Sparse Random Linear Network Coding

The computation overhead involved in the content distribution consists of two parts. The first part is the cost due to the verification of the packets, and the second part is the due to the need to compute random combinations of the data blocks. The preceding sections of this paper focus on the first part of the cost, which can be reduced through the use of more efficient hash functions and batch verification techniques as we have discussed. Nevertheless, the second part of the cost also plays a very important role in practice, especially when the content is large (e.g., in the order of gigabytes), and it has a significant impact on the choice of parameters.

In some previous work (such as [2], [3]), it is proposed to divide the content to be distributed into smaller trunks (sometimes referred to as *generations*), and random linear network coding is applied to each trunk of content independently. Although this method works in certain application scenarios, it does not address the problem directly but instead avoids high computation overhead by applying random linear network coding to smaller problem instances. Hence, this strategy may lose certain benefits from network coding. For example, when a node sends data to its downstream nodes, it has to decide which trunk to send. If the algorithm to make such decisions is not designed properly, it may result in a situation where a certain trunk cannot be reconstructed after a few key nodes

have left the network.

Here we propose a simple yet powerful alternative to avoid high computation cost when computing the random combinations. We will refer to this method as *Sparse Random Linear Network Coding*. The idea is that, instead of computing a random combination of all the n data blocks, we can instead randomly select only θ of them and compute a random combination of only those θ blocks. More precisely, when a node A needs to send a packet (\mathbf{x}, \mathbf{c}) to its downstream node, it performs the following steps.

SPARSE RANDOM LINEAR NETWORK CODING

- 1) Randomly choose θ packets from the random combinations received by A so far. Let these packets be $(\mathbf{x}_1, \mathbf{c}_1), \dots, (\mathbf{x}_\theta, \mathbf{c}_\theta)$.
- 2) Randomly choose $r_1, \dots, r_\theta \in \mathbb{Z}_q$.
- 3) Compute packet (\mathbf{x}, \mathbf{c}) as

$$\mathbf{x} = \sum_{i=1}^{\theta} r_i \mathbf{x}_i, \quad \mathbf{c} = \sum_{i=1}^{\theta} r_i \mathbf{c}_i.$$

Also, we require the source node to be more powerful than other nodes and still sends random combinations of all n blocks. It is clear that each packet being sent over the network would still be quite random, and allow high probability of reconstruction at the receivers. We confirm this intuition through experiments in Section VIII-B.

It is worth the note that the probability of successful delivery is very high even with small constant θ . Therefore, the computation overhead due to the computation of random combinations can be made independent of the number of blocks, which greatly relaxes the constraints that need to be considered for practical systems.

In addition, the sparse coding variant is just as secure as the basic scheme. The proof of security of the basic scheme can be applied for sparse coding without modifications, since the proof does not require the coefficients to be of any special properties.

VII. Communication Overhead

The communication overhead of the content delivery scheme involves three parts, namely the cost of distributing the parameters, the hash values, and the random coefficients used in the combinations.

A. Parameters

The first part is the cost of distributing the parameters for the delivery. Here we are mainly concerned about the parameters related to the hash function, which includes n the number of blocks, m the number of subblocks per block, and primes p

and q . Since the algorithm to find the small prime bases p_i 's is public and deterministic, all nodes in the network can compute those p_i 's locally after receiving the above parameters. Note that the first part of the overhead is independent of the content size, and can be regarded as constant. On the other hand, if the scheme proposed in [14] is used instead, this part of the overhead would be at least the same size of a data block, which may be significant, since the number of blocks n cannot be too large (as we will show later in Section VII-C).

B. Hash Values

The second part is the cost of distributing the hash values of the data blocks before the actual delivery of the data. Since each hash value is an element in \mathbb{Z}_p , the total size of the hash values is $n|p| = n\lambda$. This part of the cost is proportional to the number of data blocks, and it is a one-time cost for any node in the network for the entire content distribution session. The ratio of the cost over the total size of content is $\lambda/(m\gamma)$, which is typically very small. For instance, $\lambda = |p|$ is typically at most 2k bits, and $m\gamma = m|q|$ is roughly the size of one data block, which is typically at least 32k bytes. In this case the overhead ratio is about 0.78%. When the data blocks are of a few megabytes in size, as suggested in some of the previous work, such overhead would be quite insignificant.

C. Coefficients

Recall that each packet transmitted over the network consists of a pair (\mathbf{x}, \mathbf{c}) , hence the overhead due to the need to transmit the coefficient vector \mathbf{c} has to be examined closely. Since each coefficient is in the same finite field as each subblock of the content, the size of one coefficient is γ . Hence the total size of the coefficient vector is $n\gamma$. Considering that the size of each data block is $m\gamma$, the overhead ratio due to coefficient vectors is n/m .

Since this part of the overhead is re-occurring in the sense that it will happen each time some data is transmitted, it is desirable to make the ratio n/m as small as possible. Hence, as we mentioned earlier, n can not be too large. For instance, if the size of the data is 100 megabytes, $\gamma = 800$ (say, with $|p| = \lambda = 1024$), and we want to make $n/m < 0.01$, we must have $n \leq 2^{10}/10 \approx 100$. When the equality holds, $m = 100n = 10 \times 2^{10} \approx 10000$ and each block is of size 1 megabyte. If the original data is of size 1 gigabyte, with the same γ , roughly we can have at most 320 blocks of 3.2 megabytes each.

For dynamic networks where nodes frequently join and leave the network, or in scenarios where nodes have slow connections to each other, it may be desirable to have data blocks as small as possible. One way to achieve that is to reduce γ , at the cost of reduced security. For example, when $\gamma = 400$ (say, with $|p| = \lambda = 512$), and $n/m < 0.01$, we have $n \leq 144$ for 100 megabyte data, and each block is at least about 700 kilobytes.

D. Trade-offs

Besides trade-offs within the two categories of overhead, we can actually trade-off between the communication and computation overhead.

The main idea is that we can divide the original content \mathbf{X} into smaller trunks and re-use the coefficient vector for different trunks. This is similar to some previous work (e.g., [2], [3]), but the difference is that we divide \mathbf{X} ‘‘horizontally’’ (versus ‘‘vertically’’ as in previous work) if we view the original content in the matrix form. In particular, each trunk contains m_t rows of \mathbf{X} , and during each transmission from a node to one of its downstream nodes, all the m/m_t trunks share the same coefficient vectors. As a result, we can send only one coefficient vector for m/m_t trunks.

In this way, we can remove the dependency between m_t and n to some extent, so that the computation overhead can be reduced when we apply the enhanced batch verification scheme in Section V-C, since now m_t and b can be made closer.

The price of doing this is twofold. First, we need to distribute hash values for each trunk independently. As a result, the one-time communication cost due to the hash values increases by a factor of m/m_t . On the other hand, during the verification, we need to compute H_2 as in Equation (6) exactly m/m_t times for a batch, instead of only one-time. As we analyzed earlier, when we require small re-occurring communication overhead, n/m should be small. Hence, such increase may be offset by the effect of the enhanced batch verification.

VIII. Experiments

A. Computation Overhead

As we mentioned in Section VI, the computation cost involves both the computation of random combinations and the hash values. When sparse random linear coding is applied, for each combined block we only need to compute the combination of θ blocks, which makes the computation of combinations much more efficient than that of the hash values. Hence we only analyze the computation cost of the hash function.

Since we obtain the efficiency by using small prime bases in the hash function, the computation of H_1 is much more efficient than H_2 as in Equation (6), when $m = n$. However, in typical cases, m may be much larger than n (say, $n < 0.01m$), hence the overall computation cost of H_1 could be more than that of H_2 .

We implement the proposed hash function and conduct experiments to evaluate the efficiency of the computation of H_1 and H_2 with various parameter combinations. We use GNU C/C++ compiler with open source Crypto++ library, and the experiments are done on a laptop computer with an Intel

Core 2 Duo T7300 CPU. Only one of the two cores is utilized during the experiments.

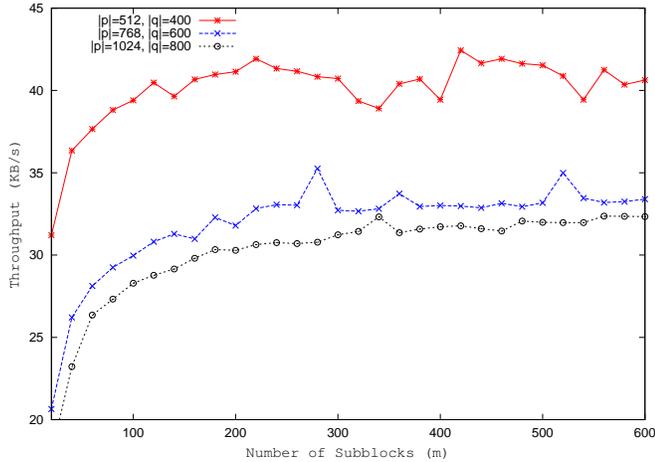


Fig. 2. Computation Efficiency of H_1 .

In Fig. 2, we show the throughput of the computation of H_1 , which is computed as the size of a data block divided by the average time it takes to compute a hash value for the block. The values are taken as the average of 25 random instances. In general, we can see that the throughput increases when the number of subblocks per block (i.e., m) is increased. Nevertheless, such increment will not be very obvious after m exceeds a certain value.

We can also see that even with relatively small values of λ and γ (say, $\lambda = 512$ and $\gamma = 400$), the computation of H_1 cannot be very efficient (about 40 KB per second) compared with common hash functions such as SHA-1. With carefully designed pre-computation methods (such as that in [14]), the throughput can be increased by a small constant factor r at the price of a storage requirement that is 2^r times that without pre-computation. Furthermore, by performing batch verification, and ignoring the cost to compute the linear combination during batch verification, the throughput can be increased by a factor of b , which is number of blocks in a batch. However, in typical applications, it is not desirable to have a too large b , since otherwise a node needs to wait for too long to receive a batch.

From these parameters, we can compute the throughput from the graph. For example, when we choose $\lambda = 1024$ and $\gamma = 800$, and $m \geq 200$, the throughput without batch verification is at least 30 kilobytes per second. With batch verification and $b = 20$, the throughput is increased to 600 kilobytes per second, and if $b = 100$, the throughput of computing H_1 alone can reach about 3 megabytes per second. With pre-computations, the throughput can be further improved by a small factor.

If we use smaller security parameters λ and γ , the overall throughput can be increased. It is interesting to see that reducing λ from 1024 to 768 does not significantly increase the throughput, but further reduction to 512 shows more than 30% of speedup (from 30 to 40 kilobytes per second).

We further evaluate the computation efficiency of H_2 during verification. The results are shown in Fig. 3. We can easily see that the time it takes to compute H_2 increases linearly with the number of blocks (i.e., n). Furthermore, the efficiency of computing H_2 is quite predictable in the sense that, when we increase the parameters (n , λ or γ), the increment of the time consumption is proportional to n and the difference in λ (or γ , since we have made λ/γ constant in our experiments).

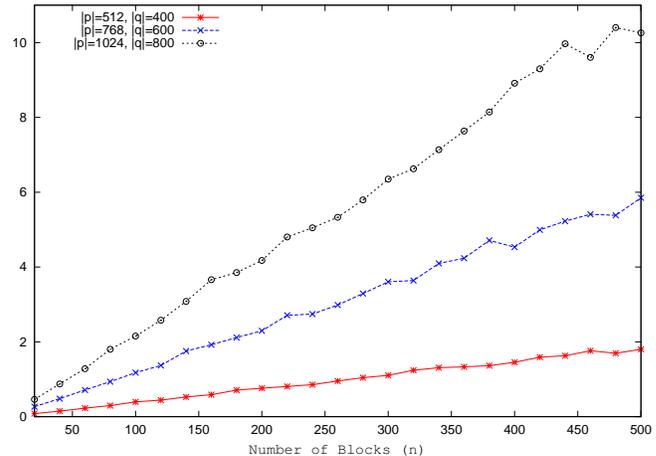


Fig. 3. Computation Efficiency of H_2 .

Interestingly, Fig. 3 can also be used as a reference to the computation efficiency of the original hash function proposed in [14]. The throughput under different security parameters are the gradient of different curves in the figure. For example, for $|p| = 512$ and $|q| = 400$, when $n = 500$, the time it takes to compute H_2 is about 2 seconds. This translates to a throughput of about 12.5 kilobytes per second, which is much lower than 40 kilobytes per second as in our proposed scheme. For $|p| = 1024$ and $|q| = 800$, similar calculations show that the throughput is greatly reduced to about 5 kilobytes per second, whereas in our scheme the throughput is only reduced to 30 kilobytes per second. Hence, we can see that our scheme has the advantage that the efficiency is not reduced as much when the parameters are increased for stronger security. The computational advantage of our scheme is mainly due to the use of deterministically chosen small primes as the bases for exponentiations, which is the rationale behind the design of the VSH scheme ([17]).

From Fig. 2 and Fig. 3, we can compute the final throughput of the hash verification with given parameters. For example, using numerical examples in Section VII-C, if the original content size is 100 megabytes, $m = 10 \times 2^{10}$, $n = 100$, $|p| = \lambda = 1024$ and $|q| = \gamma = 800$, the throughput of computing H_1 is at least 30 kilobytes per second, and the time it takes to compute H_2 is about 2 seconds. Hence, the overall time it takes to verify a packet is roughly 36 seconds, which translates to an overall throughput of about 28 kilobytes per second. If $b = 20$, this throughput would be increased by a factor of 20, which gives 560 kilobytes per second. For the same λ and γ , when the size of the original data is 1 gigabyte

with $n = 320$ and $m = 33555$ (which gives a block size of about 3.2 megabytes), similar calculation shows that the overall throughput without batch verification is also about 28 kilobytes per second.

B. Effectiveness of Sparse Random Linear Network Coding

To evaluate the effectiveness of the sparse random linear network coding as described in Section VI, we conduct the following experiments.

First, we build a random directed graph with N nodes, where we select a random root node R . We randomly select the edges such that each node has at least k incoming edges. In this way, we can be sure that the min-cut from R to any other node is at least k . Furthermore, this is similar to bit-torrent-like peer-to-peer networks, where each peer randomly selects a number of nodes as its upstream nodes.

Next, we assume that R has n blocks of data to be distributed to all other nodes in the network. The distribution is done one step at a time. In each step, for all the nodes that have received data from all of their upstream nodes generate their own combinations and deliver them to their downstream nodes. This process is repeated until no further delivery is possible.

Lastly, we examine the data received by each node, and determine if it is sufficient for the node to reconstruct the original data. The effectiveness of the coding scheme is measured by the percentage of receiving nodes that can successfully reconstruct the data at the end of the experiments.

In our experiments, we choose $n = 120$, and vary k and N to examine the reconstruction probability. We choose $q = 251$, which is relatively small. Generally speaking, a larger q would give larger reconstruction probability. However, as we can see later, such small q already yields high reconstruction probabilities. Furthermore, we choose $\theta = 2$, which means we only need to compute the combination of two randomly selected packets each time. The results of our experiments are summarized as the table below.

	$k = 20$	$k = 40$	$k = 60$
$N = 200$	0.9950	0.9950	1
$N = 400$	0.9975	0.9975	0.9975
$N = 600$	0.9866	0.9950	0.9917
$N = 800$	0.9962	0.9975	0.9850
$N = 1000$	0.9950	0.9890	0.9950

TABLE II
RECONSTRUCTION PROBABILITIES

As we can see from Table II, the reconstruction probabilities are very close to 1, which shows that the sparse variant of the random linear coding performs just as good as the original random linear coding even for very small θ .

IX. Conclusions

Researchers have shown successful application of network coding in wireless networks [23], [24] to improve system throughput, or P2P networks to improve overall system efficiency. In this paper, we investigate the security and efficiency issues in large content distribution based on network coding.

We consider the problem of on-the-fly verification of the integrity of the data in transit. Although a previous scheme based on homomorphic hash functions is applicable, it was mainly designed for server side coding only, and will be much less efficient when it is applied on random network coding. We propose a new on-the-fly verification scheme based on a faster homomorphic hash function, and proved its security.

We also consider the computation and communication cost incurred during the content distribution process. We identify various sources of the cost, and investigate ways to eliminate or reduce the cost. In particular, we propose a sparse variant of the classical random linear network coding, where only a small constant number of blocks are combined each time. Furthermore, we discuss some possible enhancements under certain conditions of the parameters, and ways to trade-off among different cost.

Experiments are conducted to examine the efficiency of the proposed hash function, as well as the effectiveness of the proposed sparse random linear network coding. The results show that the new hash function is able to achieve reasonable speed, and the sparse variant performs just as well as the random network coding using typical parameters.

Acknowledgement: the work of John C.S. Lui was supported in part by the RGC Project 415309.

REFERENCES

- [1] S. Acedanski, S. Deb, M. Medard, and R. Koetter, "How good is random linear coding based distributed networked storage," in *Workshop on Network Coding, Theory and Applications*, Italy, April 2005.
- [2] P. A. Chou, Y. Wu, and K. Jain, "Practical network coding," in *Allerton Conf. Commun., Contr., and Comput.*, October 2003, invited paper.
- [3] C. Gkantsidis and P. R. Rodriguez, "Network coding for large scale content distribution," in *IEEE INFOCOM*, 2005, pp. 2235–2245.
- [4] M. Wang, Z. Li, and B. Li, "A high-throughput overlay multicast infrastructure with network coding," in *IWQoS*, 2005.
- [5] Y. Zhu, B. Li, and J. Guo, "Multicast with network coding in application-layer overlay networks," *IEEE JSAC*, vol. 22, pp. 107–120, 2004.
- [6] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung, "Network information flow," *IEEE Trans. Inf. Theory*, vol. 46, no. 4, pp. 1204–1216, 2000.
- [7] S. R. Li, R. W. Yeung, and N. Cai, "Linear network coding," *IEEE Trans. Inf. Theory*, vol. 49, no. 2, pp. 371–381, 2003.
- [8] R. Koetter and M. Médard, "An algebraic approach to network coding," *IEEE/ACM Trans. Networking*, vol. 11, no. 5, pp. 782–795, 2003.
- [9] S. Jaggi, P. Sanders, P. A. Chou, M. Effros, S. Egner, K. Jain, and L. M. Tolhuizen, "Polynomial time algorithms for multicast network code construction," *IEEE Trans. Inf. Theory*, vol. 51, no. 6, pp. 1973–1982, June 2005.
- [10] C. Gkantsidis and P. Rodriguez, "Cooperative security for network coding file distribution," Microsoft Research, Tech. Rep., 2004.
- [11] T. Ho, B. Leong, R. Koetter, M. Médard, M. Effros, and D. R. Karger, "Byzantine modification detection in multicast networks using randomized network coding," in *IEEE Intl. Symp. Inf. Theory*, 2004.

- [12] C. Gkantsidis, J. Miller, and P. Rodriguez, "Anatomy of a P2P content distribution system with network coding," in *Intl. Workshop on Peer-to-Peer Systems*, Santa Barbara, CA, February 2006.
- [13] T. Ho, R. Koetter, M. Médard, D. R. Karger, and M. Effros, "The benefits of coding over routing in a randomized setting," in *IEEE Intl. Symp. Inf. Theory*, 2003.
- [14] M. N. Krohn, M. J. Freedman, and D. Mazières, "On-the-fly verification of rateless erasure codes for efficient content distribution," in *IEEE Symp. Security and Privacy*, Oakland, CA, May 2004, pp. 226–240.
- [15] M. Bellare, O. Goldreich, and S. Goldwasser, "Incremental cryptography: The case of hashing and signing," in *CRYPTO*, 1994.
- [16] C. Gkantsidis and P. Rodriguez, "Cooperative security for network coding file distribution," in *IEEE Infocom*, April 2006, pp. 1–13.
- [17] S. Contini, A. K. Lenstra, and R. Steinfeld, "VSH, an efficient and provable collision-resistant hash function," in *Eurocrypt*, ser. LNCS, vol. 4004, 2006, pp. 165–182.
- [18] R. Koetter and M. Médard, "Beyond routing: An algebraic approach to network coding," in *IEEE INFOCOM*, 2002, pp. 122–130.
- [19] J. Edmonds, "Minimum partition of a matroid into independent sets," *J. Res. Nat. Bur. Standards Sect.*, vol. 869, pp. 67–72, 1965.
- [20] B. Fan, J. C. S. Lui, and D.-M. Chiu, "The design trade-offs of bittorrent-like file sharing protocols," *IEEE/ACM Trans. Netw.*, vol. 17, no. 2, pp. 365–376, 2009.
- [21] R. T. B. Ma, S. C. M. Lee, J. C. S. Lui, and D. K. Y. Yau, "Incentive and service differentiation in p2p networks: a game theoretic approach," *IEEE/ACM Trans. Netw.*, vol. 14, no. 5, pp. 978–991, 2006.
- [22] M. Bellare, J. Garay, and T. Rabin, "Fast batch verification for modular exponentiation and digital signatures," in *Eurocrypt*, 1998.
- [23] J. Le, J. C. Lui, and D.-M. Chiu, "On the performance bounds of practical wireless network coding," *IEEE Transactions on Mobile Computing*, vol. 9, pp. 1134–1146, 2010.
- [24] J. Le, J. C. S. Lui, and D.-M. Chiu, "Dcar: Distributed coding-aware routing in wireless networks," *IEEE Transactions on Mobile Computing*, vol. 9, no. 4, pp. 596–608, 2010.



Qiming Li received B. Eng (Computer Engineering) and Ph.D from the Department of Computer Science, School of Computing, National University of Singapore (NUS). From 2006 to 2007, he was a post-doc research fellow in Polytechnic University with Professor Nasir Memon. His research interests include cryptography, multimedia security, network security and algorithms.



John C.S. Lui received his Ph.D. in Computer Science from UCLA. He is currently the chairman of the Computer Science & Engineering Department at The Chinese University of Hong Kong. His current research interests are in data networks, system and applied security, multimedia systems, online social networks and cloud computing. John received various departmental teaching awards and the CUHK Vice-Chancellor's Exemplary Teaching Award, as well as the co-recipient of the Best Student Paper Awards in the IFIP WG 7.3 Performance 2005 and

the IEEE/IFIP Network Operations and Management (NOMS) Conference. He is an associate editor in the Performance Evaluation Journal, IEEE-TC, IEEE-TPDS and IEEE/ACM Transactions on Networking. John is a Fellow in ACM and IEEE.



Dah-Ming Chiu received the B.Sc degree in Electrical Engineering from Imperial Collage, University of London, and the Ph.D degree from Harvard University, in 1975 and 1980 respectively.

He was a Member of Technical Staff with Bell Labs from 1979 to 1980. From 1980 to 1996, he was a Principal Engineer, and later a Consulting Engineer at Digital Equipment Corporation. From 1996 to 2002, he was with Sun Microsystems Research Labs. Currently, he is a professor and the Chairman in the Department of Information Engineering in The Chinese University of Hong Kong. He is known for his contribution in studying network congestion control as a resource allocation problem, the fairness index, and analyzing a distributed algorithm (AIMD) that became the basis for the congestion control algorithm in the Internet. His current research interests include economic issues in networking, P2P networks, network traffic monitoring and analysis, and resource allocation and congestion control for the Internet with expanding services. Two recent papers he co-authored with students have won best student paper awards from the IFIP Performance Conference and the IEEE NOMS Conference. Recently, Dr Chiu has served on the TPC of IEEE Infocom, IWQoS and various other conferences. He is a member of the editorial board of the IEEE/ACM Transactions on Networking, and the International Journal of Communication Systems (Wiley).