Performance Evaluation of Secure Network Coding using Homomorphic Signature

Seung-Hoon Lee*, Mario Gerla*, Hugo Krawczyk†, Kang-Won Lee†, Elizabeth A. Quaglia‡
*Dept. of Computer Science, University of California, Los Angeles
†IBM T.J. Watson Research Center
‡Royal Holloway, University of London
{shlee,gerla}@cs.ucla.edu, {hugokraw, kangwon}@us.ibm.com, E.A.Quaglia@rhul.ac.uk

***Abstract*—Network coding has gained significant attention by improving throughput and reliability in disruptive MANETs. Yet, it is vulnerable to attacks from malicious nodes. In order to prevent malicious attacks, we have explored the use of secure network coding schemes based on homomorphic properties of cryptographic systems. While homomorphic methods, especially those based on public key cryptography, provide strong protection against both external and internal attacks, they do increase processing overhead as they require complex cryptographic operations (e.g., exponentiation, multiplication, modular operations). The goal of this paper is two fold: assess the feasibility of implementing Homomorphic Network Coding in an off-the-shelf laptop/smartphone platform, and evaluate the processing and delay performance when such implementations are deployed in a simple network scenario. To this end, we have implemented in LINUX an RSA-based homomorphic algorithm which exhibits very competitive processing efficiency as compared with published (public-key) schemes. For the LINUX implementation we have measured the processing delay for various flow and parameter settings. We have then integrated the homomorphic processing model (with associated O/H) in a MANET network simulator. Using this simulator, we have evaluated the performance of secure network coding under various network conditions and have compared it with other secure network coding schemes. We conclude the paper with a discussion of secure coding feasibility and cost for different application scenarios.**

I. Introduction

Network coding [3], [20] provides an alternative technique to traditional multicast routing which improves network throughput and reliability. We consider a network with a source node that intends to share a file with multiple receivers. Conventionally, the file is divided into multiple blocks which are transmitted by the source node to the destinations. With network coding, the source node generates encoded blocks by combining all of the blocks of the file, and then it sends out the encoded blocks instead of the original blocks. Intermediate nodes in the network also participate in the network coding operation. Namely, they further encode the blocks before forwarding them to other nodes. The encoding operation at the source and at intermediate nodes creates redundancy in the original data stream and increases network resilience; even if some of the encoded blocks are dropped under unstable wireless network conditions due, for example, to jamming or interference, the receivers are still able to recover the original data from the remaining coded blocks.

This research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the author(s) and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

Although network coding is designed to improve network performance and reliability, it is nevertheless vulnerable to *pollution attacks*. This very inexpensive type of denial of service attack can be mounted by malicious nodes by injecting bogus coded blocks in the network, therefore subverting the reconstruction of the original information and severely affecting performance. In order to prevent these attacks, information-theoretic [23], [24] and cryptographic solutions have been considered. In the latter context several *secure network coding* schemes have been proposed, both in the symmetric and public-key setting [5], [7], [12], [16], [17], [31]. The latter type, based on public-key techniques, are particularly powerful for securing network coding as they allow intermediate and destination nodes to *immediately* single out any contaminated packets without having to place any trust on any party other than the information source.

While these solutions can protect network coding from attacks, the practical aspects of secure network coding implementation have not been well investigated in the literature. These schemes require complex cryptographic computations (e.g., exponentiation, multiplication, modular operations) at each intermediate node causing significant computational overhead. Thus computation can significantly affect the application performance, especially when nodes are resource constrained (e.g., processing power, memory space, battery capacity, etc.)

The main goal of this paper is to assess the practical feasibility of cryptographic network coding techniques based on public-key cryptography which, as said, offer the most complete defense against pollution attacks. We investigate the processing overhead of secure network coding in real network environments considering heterogeneous node types and information flows and evaluate the feasibility of the implementation. First, we implement in Linux the RSA-based secure network coding schemes (based on homomorphic signature and hashing, respectively) from [12]. This choice is motivated by the fact that these schemes offer the lowest computation complexity when compared to other existing (public-key) secure network coding schemes. A thorough discussion and performance comparison with other public-key schemes is provided in VI, where we illustrate in detail how the alternative solutions, specifically the homomorphic signature scheme of [5] and the homomorphic hashing schemes originating from [17], are computationally more expensive than the RSA-based solutions from [12]. From the Linux implementation, we measure the actual processing overhead of each homomorphic operation. Using the experimental results, we integrate the homomorphic processing overhead in a network simulator, QualNet [28].

The remainder of this paper is organized as follows. We discuss related work in Section II. In Section III we give an overview on network coding and the homomorphic signature and hashing schemes from [12]. In Section IV, we present the computational models and experimental results. We evaluate the processing overhead in Section V. We compare the performance of our scheme with alternative solutions in Section VI. We conclude the paper in Section VII.

II. Related Work

The benefits of network coding have been extensively studied not only in theory [3], [20] but also at the more practical level of protocol operations required under various scenarios [6], [8], [13], [19], [27]. Network coding has been found to be effective to disseminate information in wireless networks because nodes can utilize the broadcast nature of the wireless medium to efficiently mix overheard information. Most importantly, recent results relative to streaming applications [19], [27] show that network coding offers adaptive protection against random losses caused by mobility and jamming. This is because the nodes in the critical section can locally adjust the forward redundancy according to measured random loss. The redundancy is eliminated when it is no longer necessary (say, past the critical section) by systematically removing linearly dependent blocks. This makes Network Coding much more attractive in spotty jamming attacks than other options such as end to end redundancy schemes like Fountain Codes or Raptor Codes. The latter possess only limited adaptive redundancy and furthermore require end to end feedback to exercise it.

As security concerns about network coding have emerged, several solutions to thwart pollution attacks have been proposed. Information-theoretic methods [23], [24] offer security only against a relatively limited class of adversaries and incur in a significant communication cost. Many cryptographic solutions exist, both in the symmetric and public-key setting. While symmetric key solutions are computationally more efficient than public-key ones, they suffer from significant limitations. Specifically, the "secure random checksums" technique from [14] requires parties to share secret keys and hence is insufficient to solve the pollution problem in general; the "broadcast MAC" solution from [2] is open to (sufficiently large) adversarial collusions; and the TESLA-based approach of [9] is of limited applicability as it requires strong assumptions on network connectivity and synchronization under adversarial attack. In this setting also lies another line of research whose focus is on detecting the *polluting nodes* rather than detecting *polluted packets* (see [18] and the references therein).

The public-key-based solutions can be classified in two groups: Those based on homomorphic hashing [12], [17], [31] and those based on homomorphic signatures [5], [7], [12], [16]. The former is often more efficient in terms of computation but requires the transmission of long signatures with each packet or the pre-distribution of per-generation information to intermediate and target nodes. Solutions based on homomorphic hashing should be preferred, due to their better computational performance, in settings where such pre-distribution is feasible (e.g., when nodes register to receive content or when a broadcast channel from the source is available for such pre-distribution). Homomorphic signatures, on the other hand, are somewhat more expensive computationally but do not require pre-distribution or the appending of long signatures to each packet. We note that in this paper we assess the practicality of both approaches by implementing the RSA-based homomorphic hashing and homomorphic signature schemes proposed in [12]. We chose these schemes for implementation since they offer the best computational performance among existing solutions. Indeed, computational cost is the performance bottleneck of secure network coding schemes, and hence the main resource to minimize.

Most of the prior work on secure network coding studies the security aspects in a theoretical framework, without evaluating the practical implementation details. These studies tend to overlook the fact that cryptographic operations involve high computational overhead, and may significantly degrade the performance when overlaid on top of conventional network coding. In the literature, the computational overhead of cryptographic operations has been discussed in [10], [11], [21]. In [10], Freeman et al. evaluate various cryptographic schemes (e.g., RSA, authentication with keyed-hash, confidentiality using RC5) in a testbed and study the performance implications of using cryptographic controls in performance critical systems. Lie et al. evaluate the overhead of comprehensive cryptography algorithms under limited resource constraints of nodes in wireless sensor networks [21]. Ganesan et al. investigate the computational requirements for a number of popular cryptographic algorithms. They develop methods to derive the computational overhead of embedded encryption algorithms [11]. These studies inspired us to investigate the computational overhead of the cryptographic side of secure network coding. In particular, this paper aims at assessing the feasibility of secure network coding schemes in portable platforms such as PCs and smart phones.

III. Secure Network Coding

In this section, we first review the notion of *random linear* network coding scheme, then briefly describe the secure network coding schemes presented in [12] based on homomorphic signature and hashing over groups of composite order.

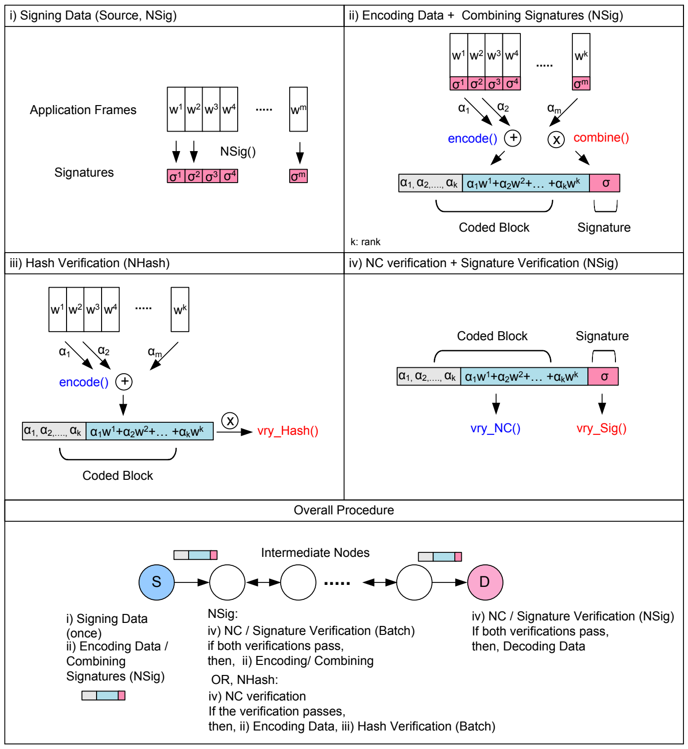

i) Signing Data (Source, NSig)
Application Frames
Signatures
NSig()
ii) Encoding Data + Combining Signatures (NSig)
encode()
combine()
Coded Block
Signature
k: rank
iii) Hash Verification (NHash)
encode()
vry_Hash()
Coded Block
iv) NC verification + Signature Verification (NSig)
Coded Block
Signature
vry_NC()
vry_Sig()
Overall Procedure
Intermediate Nodes
S
D
i) Signing Data (once)
ii) Encoding Data / Combining Signatures (NSig)
NSig:
iv) NC / Signature Verification (Batch)
if both verifications pass,
then, ii) Encoding/ Combining
OR, NHash:
iv) NC verification
If the verification passes,
then, ii) Encoding Data, iii) Hash Verification (Batch)
iv) NC / Signature Verification (NSig)
If both verifications pass,
then, Decoding Data

Fig. 1. Secure Network Coding Operations

A. Network Coding

Assume a source node $\mathbf{S}$ intends to share a file $\mathbf{F}$ with multiple receivers in the network. $\mathbf{S}$ divides the file $\mathbf{F}$ into m blocks, $\mathbf{p}_1, \mathbf{p}_2, \cdots, \mathbf{p}_m$, where the size of each block is n. Each original block $\mathbf{p}_\ell$ ($\ell = 1, \cdots, m$) has the ℓth unit vector $\mathbf{e}_\ell$, the *encoding vector*, prepended to it. The original block $\mathbf{p}_\ell$ is represented as $\mathbf{w}_\ell = [\mathbf{e}_\ell, \mathbf{p}_\ell]$. Instead of transmitting the original blocks of $\mathbf{F}$, $\mathbf{S}$ generates augmented blocks via *weighted random linear combination* of all the blocks. The coded block $\bar{w}$ is defined as $\sum_{k=1}^{m} c_k \mathbf{w}_k$, where c_k is randomly drawn over a finite field $\mathbb{F}$.

When an intermediate node receives a coded block $\bar{w}$, the node first checks linear independency of the incoming block. If the received block is linearly independent from other blocks that have been stored, the node accepts and stores the block. The total number of linearly independent coded blocks stored is called *rank*. By combining all the coded blocks collected so far, each intermediate node regenerates a coded block: $\sum_{k=1}^{rank} c_k \bar{\mathbf{w}}_k$, where c_k is also randomly chosen from $\mathbb{F}$.

In order to recover the original blocks of $\mathbf{F}$, a receiver node must collect at least m coded blocks carrying encoding vectors (c) that are linearly independent. Denote the valid coded blocks $\bar{\mathbf{w}}_\ell = [c_\ell, \bar{p}_\ell]$, ($\ell = 1, \cdots, m$). In addition, we define $\mathbf{C}$ whose rows are the vectors c_ℓ, and $\bar{\mathbf{P}}$ whose rows are the blocks $\bar{p}_\ell$. Then the original file can be decoded as $\mathbf{P} = \mathbf{C}^{-1}\bar{\mathbf{P}}$, where $\mathbf{P}$ is a matrix whose rows are the original blocks of $\mathbf{F}$.

In order to apply the signature and hashing schemes described in next sections, we change the operation of the network coding protocol to work over the ring of integers rather than over a finite field [12]. In this case, blocks are represented by vectors with integer coordinates and the random coefficients chosen to produce random linear combinations are taken from a set $\{0, \ldots, q-1\}$ where q is a small (e.g., 8-bit) prime.

B. An RSA based Network Coding Signature Scheme

In the RSA-based homomorphic scheme proposed in [12] the source generates a signature of each coded block using a secret private key, and the signature is transmitted with the coded block. Upon receiving the coded block along with a signature, nodes validate the coded block with the given RSA information using the public key that is made available via broadcast, possibly piggybacked with the coded data. Nodes only store the coded blocks that pass the verification, and use them for further mixing. The scheme is denoted as NSig, and the details are as follows.

The NSig scheme has a parameter L, which limits the number of hops that a packet can traverse. Given L, a bound B is defined as $(mq)^L$ where q is a small prime number (≈ 256) that defines the range of coefficients chosen for random linear combinations. B represents the largest possible coordinate in any block c transmitted in the network, and M denotes an upper bound on each of the coordinates of the initial blocks $\mathbf{p}_1, \cdots, \mathbf{p}_m$ transmitted by the source. Then, the maximal coordinate in a valid block v transmitted in the network is BM, and it is denoted by B^*.

Based on the parameters described, the source node $\mathbf{S}$ has a public key $(N, e, g_1, \ldots, g_n)$, where N is a product of two large safe primes, e is the public RSA exponent chosen as a prime larger than mB^*, and $g_1, \ldots, g_n$ are random generators of the cyclic subgroup $\mathbb{G}$ of quadratic residues modulo N. In addition, a private signing key d is defined such that $ed = 1 \mod \phi(N)$ $(d \leq \phi(N)$ as in regular RSA).

•**Signing data**, NSig(w)**:**

Using the given RSA keys, $\mathbf{S}$ signs a block $w = (c_1, \ldots, c_m, p_1, \ldots, p_n)$.

$$\mathsf{NSig}(w) = \left(\prod_{i=1}^{m} h_i^{c_i} \prod_{j=1}^{n} g_j^{p_j}\right)^d \mod N \qquad (1)$$

where the $h_i = H(i, \mathsf{fid}), i = 1, \ldots, m$. fid is the file identifier, thus h_i's are file specific. When $\mathbf{S}$ transmits w, the signature NSig(w) is attached with w.

•**Verification**, vry_Sig$(w, \sigma, \mathbf{S}, \mathsf{fid})$**:**

Upon receiving a block w, a node immediately discards if any of the c is negative or larger than B, or any of the $\bar{p}$ is negative or larger than B. Otherwise, the node retrieves the public key and verifies the block. w is accepted (vry_Sig$(w, \sigma, \mathbf{S}, \mathsf{fid})$ = 1) if and only if

$$\sigma^e \overset{?}{=} \prod_{i=1}^{m} h_i^{c_i} \prod_{j=1}^{n} g_j^{\bar{p}_j} \mod N \qquad (2)$$

•**Combination**, Combine$(\bar{w}_1, \ldots, \bar{w}_{rank}, \sigma_1, \ldots, \sigma_{rank})$**:**

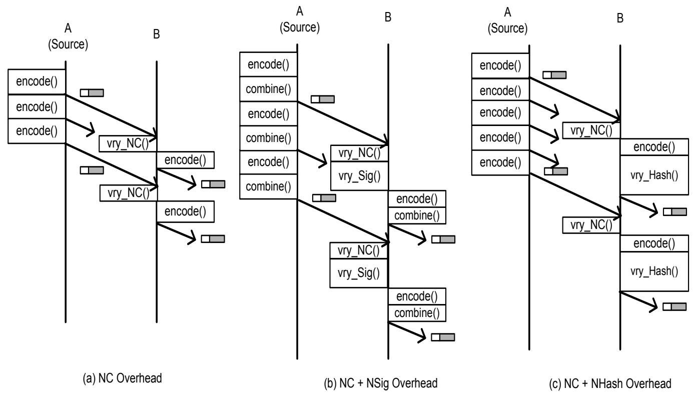
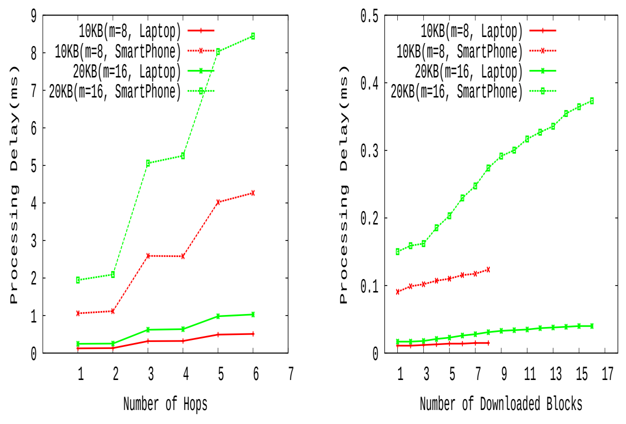


Fig. 2. Communication Models



(a) Encoding Delay (b) NC Verification Delay

Fig. 3. Processing Delay: Network Coding over the Integers

An intermediate node generates a new coded block w $(=\sum_{k=1}^{rank} c_k \bar{\mathbf{w}}_k)$ by combining all of coded blocks that passed the verification procedure. The signature of w is computed as:

$$\sigma = \prod_{i=1}^{rank} \sigma_i^{c_i} \mod N \qquad (3)$$

where $rank$ is the number of coded blocks downloaded, and c_i's are the random coefficient used to generate w.

C. Homomorphic Hashing

[12] also proposes a network coding scheme based on homomorphic hashing. Instead of verifying signatures of each coded block, nodes can validate any incoming blocks by comparing hash values, which correspond to the original blocks. This scheme can offer computational benefits compared to the homomorphic signature scheme [12] and it is denoted as $\mathbf{H}_N$. The hash function is computed with modulus N, and the packet blocks are defined over the integers. Upon receiving a coded block $w = (c_1, \ldots, c_m, \bar{p}_1, \ldots, \bar{p}_n)$, a node verifies the block as

$$\prod_{i=1}^{m} h_i^{c_i} \overset{?}{=} \mathbf{H_N} = \prod_{j=1}^{n} g_j^{\bar{p}_j} \mod N \qquad (4)$$

where $h_1, \ldots, h_m$ are the hash values of the original blocks of $\mathbf{F}$.

IV. Computation & Communication Models

In this section, we present the models of secure network coding schemes. We assume each node has memory space large enough to perform the network coding and the homomorphic operations; we do not consider disk I/O overhead. Source node generates coded blocks from the original blocks of file at application layer and executes extra operations according to the homomorphic signature or homomorphic hashing scheme.

•**Network Coding:** Regardless of the homomorphic schemes used, network coding incurs computational overhead when a node encodes a block and verifies an encoded block. Fig. 2(a) represents the communication procedure of network coding. When node A has blocks of a file to propagate, it starts *encode()* operation to generate a new coded block. After generating a coded block, A transmits the coded block to B. Upon receiving the coded block, B first checks linear dependency of the block with other blocks by running *vry_NC()*. If the block is independent from the other blocks, B stores the block and uses it for further encoding.

•**Homomorphic Signature:** The signature scheme requires two additional procedures. After a node A generates a new coded block, it computes a signature of the block by *combine()*. Once both *encode()* and *combine()* procedure are finished, the newly generated coded block with the signature is ready to be sent to B. Upon receiving the coded block and signature, B first checks linear dependency. If the block is independent of other blocks, B checks the validity of the signature of the block by running *vry_Sig()*. Note that we verify the linear independence before the signature verification procedure. This is because *vry_Sig()* requires more computation than *vry_NC()* (as shown in Section V-A). Thus, if an incoming block were linearly dependent of other blocks, we can immediately discard the block after running *vry_NC()*.If the block passes both *vry_NC()* and *vry_Sig()*, B stores the block with the signature to serve other neighboring nodes.

•**Homomorphic Hashing:** Unlike the homomorphic signature scheme, the hashing method verifies hash values after generating a new coded block. After A finishes *encode()*, it verifies the hash value by invoking *vry_Hash()*. If it fails, it means that some of blocks were falsified, and the generated block must be dropped. If *vry_Hash()* succeeds, A transmits the block to B. B in turn encodes the block from A with other blocks in its memory and verifies the coded block before transmission to C.

•**Batch Verification:** The computational cost of NSig and NHash verification can be amortized by verifying groups of blocks rather than individual ones. Specifically, it is shown in [12] that in order to validate an Nsig signature on a set S of blocks it is sufficient to generate a random linear combination

	10KB(m=8)	20KB(m=16)
combine()	0.16ms	0.31ms
vry_Sig()	22.17ms	22.38ms
vry_Hash()	17.44ms	17.55ms

TABLE I
PROCESSING DELAY: SECURE NETWORK CODING

of these blocks and only verify the signature on the resultant combined block. If this verification succeeds then with high probability all blocks in S were valid. The same holds for Nhash verification.

V. EVALUATION

In this section, we first measure the processing overhead of each secure network coding operation via testbed experiments. With these results, we evaluate by simulation the performance of secure network coding in realistic network scenarios.

A. Processing Delay Experiments

In this section we first report the processing delays for the basic random linear network coding operations described in Section III-A. We then consider the delay incurred when performing the secure network coding operations required by the homomorphic functions in Section III-B and Section III-C. In order to handle the large integers of the homomorphic schemes, we use the GNU Multiple Precision Arithmetic Library [15]. Based on the implementation, we measure the processing delay of each operation on a Linux platform with an Intel Core 2 Duo T9600 processor (2.80GHz, 6MB Cache) and an Android platform with an ARM 11 processor (528MHz). We use generations[1] of size 10KB/20KB with the number of blocks being 8/16 respectively, and we set the size of the RSA modulus N to be 512 bits.

•**Network Coding:** Network coding consists of two functions, *encode()* and *vry_NC()*. *encode()* generates a coded block at the source and at intermediate nodes. A source node combines all the blocks in a generation with random coefficients. Intermediate nodes re-encode the coded blocks that they download. Fig. 3(a) shows the processing delay of the encoding operation at different hops. Since in our case network coding operates over the integers, the coding coefficients grow at each hop and consequently the computational delay on *encode()* increases with the number of hops. *encode()* delay increases from 0.12ms at source to 0.51ms at an intermediate node (6th hop). In addition, the overhead is also proportional to the number of blocks to be encoded as shown in Fig 3(a).

After receiving a coded block, the nodes verify linear dependency of the block with the other blocks they previously received by running *vry_NC()*. It turns out that the verification processing overhead also depends on the number of blocks downloaded. In Fig. 3(b), we measure the verification delay at a receiver node while downloading m coded blocks. The delay increases with the number of downloaded coded blocks. We maintain a $m \times m$ matrix of the coding coefficients of downloaded coded blocks and perform Gaussian elimination over the integers (see the Appendix). The more downloaded blocks a node has, the more processing time Gaussian elimination will require.

In Fig. 3, we also present the processing delays of each operation on different platforms: Linux platform on a laptop and Android platform on a smartphone. Due to the limited processing capability on smartphone environments, the delays of encoding and network coding verification operations on a smartphone are around 9.2 times greater than the delays on the laptop environment. The comparison of encoding and linear dependency verification delays reveals an interesting property: linear dependency checks are one order of magnitude cheaper than encoding operations. This property has a very important implication on the deployment of hybrid networks. In such networks, the powerful nodes use secure network coding, while the lightweight nodes, such as cellular phones, simply forward packets intact; yet, they detect and eliminate duplicates via linear dependency checks [26].

•**Homomorphic Signature and Hashing:** The homomorphic signature scheme is implemented using two functions, *combine()* and *vry_Sig()*. First, *combine()* generates a new signature by combining all the signatures of coded blocks. We assume that the signatures of the original blocks of the file are pre-computed since signature generation is required only once, at a source node. The homomorphic processing delay experiments were carried out for the Linux Platform. We measure the delay of *combine()* at a source node combining m signatures. Table. I shows that the processing delay naturally increases with the number of blocks to combine: the processing delay is 0.1645 ms with 8 blocks and it increases to 0.3188 ms with 16 blocks.

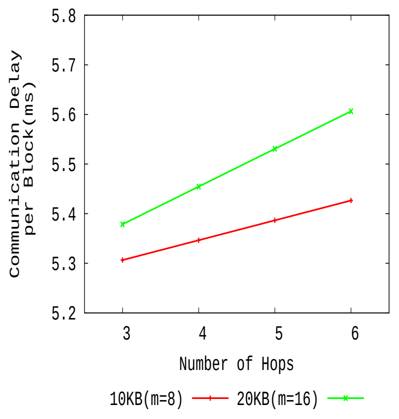
Upon receiving a coded block with a signature, intermediate nodes or destination nodes verify the signature by running *vry_Sig()* (with a batch verification, if necessary). We measure the processing overhead at a receiver node and average the delay while the node downloads the file. We present the verification delay in Table I, noting that the delays are mainly determined by the symbol size and by the number of blocks in a generation.

The homomorphic hashing scheme does a verification by checking *vry_Hash()* (with a batch verification, if necessary). Similar to *vry_Sig()*, the computational complexity of this operation depends on both the total number of blocks and the symbol size as described in Equation 4.

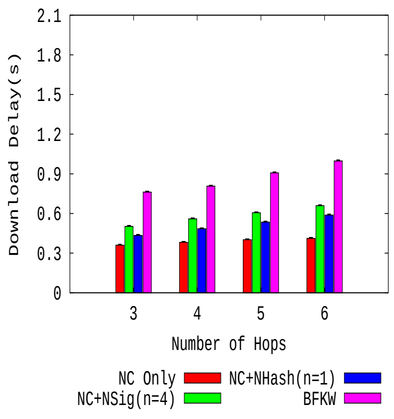
Note that the processing overheads of *vry_Sig()* and *vry_Hash()* are not proportional to the number of downloaded blocks. This is because we only validate the integrity of the incoming block according to Eq.(1) and Eq.(2) respectively.

The above processing incurs in significant overheads relative to plain network coding. This is due to the processing over the integers (that induces growing coordinates) but more significantly by the cost of cryptographic operations. In the next section we discuss the impact of this increased processing cost on network performance.

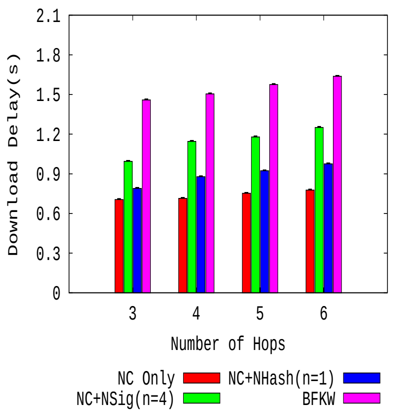
[1]A generation is a group of blocks of an application file, and we apply secure network coding to each generation.



(a) Communication Delay



(b) Download Delay (10KB, m=8)



(c) Download Delay (20KB, m=16)

Fig. 4. Impact of Hop Counts

B. Simulation

1) Simulation Setting: We implemented random linear network coding with the homomorphic signature and hashing schemes in QualNet simulator [28]. We used the experimental results on processing overhead derived in Section V-A, and incorporated them in the simulator. The test application is a multimedia unicast stream. In our simulation experiments, the source node applies homomorphic network coding to the stream. It segments the multimedia file in generations of size 10KB (m=8) and 20KB (m=16) respectively. In addition, we set the RSA modulus N to be 512 bits. The network topology (see Fig. 5) is a grid structure with multiple paths so that coded packets can be mixed at intermediate nodes. The distance between two nodes in the grid is 200m; 802.11b radio range is 350m. The link data rate in broadcast mode is 2Mbps. The source generates the multimedia stream at a rate equal to 256Kbps. We vary the number of hops between the source node and destination to evaluate the impact of path hop count on the overall performance. For each configuration, we report the value averaged over 100 runs with 95% confidence interval.

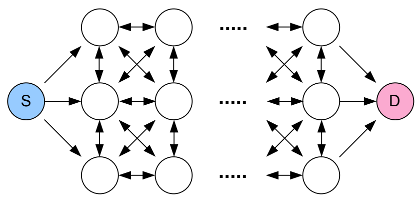


Fig. 5. Network Topology

2) Simulation Results: First of all, we investigate the communication delay increase over various hop counts. As discussed, integer field is not a closed field, thus the size of operand grows linearly with more hops. In Fig. 4(a), we present the communication delay per coded block at different hops. For example, it takes 5.34ms(m=8) for a third-hop node to send a coded block to a fourth-hop node. The growth of operands increases the size of coded blocks and consequently the communication delay increases. Note that the coded block size increases more rapidly with more blocks to be encoded in each generation; when we have 16 blocks per generation, the coefficient overhead increases 16B per hop whereas the overhead increases 8B per hop with 8 blocks per generation.

In Fig. 4(b) and Fig. 4(c) we integrate both communication and computational overhead together. The main performance metric is the *generation download delay* which is defined as the elapsed time from the start of transmission at the source until the destination node finishes downloading the generation. In addition, we compare the results of the homomorphic signature(NSig) and hashing(NHash) schemes with two reference cases, 'NC only' and 'BFKW'. In 'NC only', we only take into account the processing delay introduced by network coding. We also compare the performance of our schemes with an alternative homomorphic signature scheme, named BFKW, that was proposed in [5]. The detailed performance comparison of our scheme with BFKW is explained in VI

Fig. 4(b) and Fig. 4(c) show that the homomorphic signature scheme requires more processing to generate the new signature for each coded block, and it incurs a greater download delay compared to the vanilla network coding case. Moreover, we note that the download delay of *NSig* is larger than the delay for *NHash*. This is because *NSig* requires more operations than *NHash*. In *NSig*, sender nodes require to compute a new signature after generating a coded block, and receivers need to verify signatures of incoming blocks. In *NHash*, however, only sender nodes perform verification after creating a coded block.

In each evaluation, as we increase the number of hops, the download delay also increases, as expected, since nodes at each hop recompute the signatures, and signature delay accumulates with each hop. The homomorphic hashing scheme shows an identical pattern. In Fig. 4(b), the generation size is set to 10KB and has 8 blocks, and; in Fig. 4(c), the generation size is 20KB and composed of 16 blocks. In both cases, the size of each block is 10Kbits. We set the symbol size of each block to 10Kbits (NHash case, n=1) and 2560bits (NSig case n=4). As the destination node needs to download more coded blocks with a generation of size 20KB, the delay in Fig. 4(c) is higher.

VI. Comparing performance of alternative schemes

To assess the practical feasibility of cryptographic network coding in the public-key setting we implemented the RSA-based schemes from [12]. We now justify this choice by illustrating their performance superiority when compared to alternative schemes, namely, the homomorphic signature scheme of [5] and the homomorphic hashing schemes originating from [17].

We will base our comparison primarily on computational costs, especially the verification process, as this involves a number of costly exponentiations whose combined exponents are as long as the total length of a coded packet, and hence dominate the computation. Our evaluation of these schemes is based on currently available implementation data. While this information changes on the basis of computing platforms and specific optimizations, we believe that the following cost analysis remains valid for typical settings. As a basis for performance data we use `openssl`, the most widely available cryptographic library (underlying SSL and web application security). We also use performance data obtained using the MIRACL library [1].

Note: The small-coefficients technique introduced in [12] can be applied to the original schemes from [5] and [17] resulting in a significant improvement of these schemes (in particular, it achieves a 20-fold speed-up of signature computation at intermediate nodes). We use this optimized variant as the basis for our comparison (without this optimization, the superiority of the RSA-based solutions from [12] is even greater). We also assume an optimized version of [5] where batch verification (similar to the case of Nsig) is supported.

BFKW. We first consider [5] (to which we refer as the BFKW solution). This scheme requires to work on a pairing-friendly elliptic curve and its performance depends on three type of (costly) operations: pairings (two are required per verification), hashing into the curve (m of these are required, each of which has a cost similar to a single exponentiation), and repeated exponentiations totaling an exponent size equal to the length of a full coded packet (or vector). The latter exponentiations are similar to the right-hand side of equation (2) but in the case of BFKW this is done over many generators. Selecting a curve that optimizes all these operations at the required level of security is a complex task. We note, however, that the dominant cost is provided by signature verification at intermediate nodes and hence this is the operation to optimize. In this case, the dominant factor is the repeated exponentiations. Pairings are a costly operation[2] but except for very short vectors the long repeated exponentiations will take most of the computation time. Therefore, we judge the BFKW complexity by the cost of long exponentiation.

For a fair comparison to 512-bit RSA modulus, as used in the implementation of [12] in this paper, we will consider 112-bit elliptic curves (both choices provide a similar level of security). We first note that the advantages of elliptic curve cryptography (ECC) over RSA-based cryptography are obvious at high levels of security, but decrease as the security level gets lower. Moreover, a typical advantage of elliptic curves is the use of shorter exponents; e.g., 112-bit exponents in the ECC case vs. 512-bit exponents in the RSA case. However, in the network coding application the exponent length is independent of the underlying cryptographic primitive and is solely determined by the bit-size of packets. With longer exponents, RSA exponentiation (at the above levels of security) become significantly faster than in the ECC case.

As a concrete example, running a 112-bit curve in `openssl`, shows that an exponentiation with a 1024-bit exponent in this curve requires 7.79 ms. For exponents of the same length, RSA-512 takes just 3.2 ms. We hence have that exponentiation in RSA is faster by a factor of 2.43 which directly translates into a similar speed-up factor when going from BFKW to the RSA-based Nsig solution (the performance advantage is even larger if one considers the pairings and hashing costs proper to the BFKW solution). These numbers were obtained running `openssl` in the 32-bit implementation used in this paper's experiments. In a 64-bit implementation (in a similar machine), one gets 1.71 ms for a 1024-bit exponentiation in a 112-bit curve and 0.57 ms for a 1024-bit exponentiation in RSA-512, thus resulting in a 3 factor better performance for the RSA-based Nsig scheme. As another data point using a different ECC implementation, we have extrapolated figures obtained for 160, 224 and 256-bit curves using the MIRACL library [1], [29] to estimate the running time of a 112-bit curve on a 64-bit machine (MIRACL does not have an implementation of 112-bit curves). In this case the cost of a 1024-bit exponentiation in the 112-bit curve is 2.83 ms while with RSA (on the same machine) this time is just 0.88 ms, showing that the Nsig solution is superior to the BFKW scheme by a 3.21 factor, a similar conclusion as with `openssl`.

Homomorphic Hashing. We now consider the homomorphic hashing scheme from [17] which we refer to as EHH (exponential homomorphic hash), and how it compares to $\mathbf{H}_N$ (Section III-C). In this case there is no signature computation by intermediate nodes, just verification of incoming vectors based on the hash values. Verification in the original EHH scheme of [17] is similar to equation (4) except that the operations are done in a group of prime order with generators taken from that group. The natural instantiations in this case is

[2]In a highly pairing-optimized curve such as the BN curves [4], pairings cost roughly as much as 20 exponentiations.

to choose these groups as subgroups of $\mathbb{Z}_p^*$ or as elliptic curve groups. To match our security level one would choose p to be of 512-bit size and the elliptic curve of size 112. The former case has a computational cost similar to $\mathbf{H}_N$, as described in equation (4), for 512-bit N (since in both cases the modular multiplications have the same cost and the exponents in both cases are as the total length of the coded blocks). However, $\mathbf{H}_N$ allows for an implementation where the right-hand side of equation (4) uses a single generator fixed to 2 (see [12]) while the prime-order EHH requires many generators (e.g., 100 112-bit generators for 10,000 information bits in a typical Ethernet-size block). The reduction in number of generators results in a smaller public key and the fixing to 2 allows for optimizations in the computation of $\mathbf{H}_N$ (our current implementation does not yet take advantage of this optimization). When considering an implementation of EHH over 112-bit elliptic curve groups, we find this to be significantly worse than $\mathbf{H}_N$ (by factors of 2.4 to 3) for the same considerations as in the above study of the BFKW scheme. The above advantage of $\mathbf{H}_N$ regarding the fixed vs. many generators holds against the elliptic-curve case as well.

Overall, $\mathbf{H}_N$ provides substantial increased efficiency with respect to alternative solutions and thus it is more suitable for practice.

We conclude our discussion briefly considering the bandwidth consumption of the evaluated schemes. One of the motivations behind implementing network coding over the integers was to reduce the communication overhead naturally introduced by cryptographic solutions. In a nutshell, the idea is to start with small integer coefficients (as short as 8 bits) which will be linearly combined and hence grow. The advantages of this approach hold until the coefficient sizes do not exceed those of previous schemes (in particular BFKW and EHH). As analyzed in [12], using the small-coefficient techniques introduced in the paper, in both the BFKW and EHH cases (with 112-bit groups) we have that the RSA-based schemes have comparable overhead for coefficients growing up to 112 bits.

VII. Conclusion

Although network coding is an effective method to improve network throughput and reliability, it is vulnerable to pollution attacks by malicious nodes. In order to protect network coding streams from internal and external malicious attacks, several schemes have been proposed recently based on homomorphic properties of cryptosystems. In this paper, we implemented a secure network coding scheme based on public key cryptography. We have integrated and experimentally measured processing parameters into a network simulator and have evaluated the performance of various schemes under different scenarios. Our initial evaluation shows that secure network coding is feasible for a moderate (but realistic) multimedia data rate even without any specific optimizations or hardware-based processing. We believe more customized implementations can further reduce the processing overhead and can render secure network coding more attractive. Future work will explore the performance comparison between different secure network coding schemes and secure end-to-end coding (e.g., Fountain codes [22] and Raptor codes [30]) in stressed tactical MANET scenarios.

Appendix
Verifying Linear Independence over $\mathbb{Z}$

In typical implementations of network coding, when a node receives a set of incoming vectors it first discards any vectors in the set that are linearly dependent. In our case, this linearity verification needs to be carried over $\mathbb{Z}$. Naive Gaussian elimination over $\mathbb{Z}$ with large coordinates can be costly. Thus, we suggest to perform the test modulo a random prime of a given size. We have that if the incoming vectors are dependent in $\mathbb{Z}$ they will be dependent modulo any prime, while if they are independent in $\mathbb{Z}$ they will be dependent modulo a random prime p (of sufficient length) with very small probability. As we will see a 32-bit prime will suffice for any plausible network coding scenario.

Lemma 1: Let p be a random integer of length k and $u_1, \dots, u_t$ be linearly independent vectors in $\mathbb{Z}^m$. The probability that $u_1, \dots, u_t$ are linearly dependent mod p is at most $\mathrm{O}(\log(t!M^t))/2_k$ where M is the largest coordinate in $u_1, \dots, u_t$.

Proof: Let p be a prime of length k such that $u_1, \dots, u_t$ are linearly independent over $\mathbb{Z}$ but dependent mod p. Consider the $t \times m$ matrix U whose rows are $u_1, \dots, u_t$. Since U has rank t in $\mathbb{Z}$ then U contains t independent columns. Let U$'$ be the corresponding $t \times t$ submatrix of U, and denote by D the determinant of U$'$ (computed in $\mathbb{Z}$). We have that D $\neq 0$ but since U$'$ is of rank $\leq t$ - 1 mod p then det (U$'$) $= 0$ mod p, i.e., p divides D. How many primes can divide D? Let d denote the bit-length of D; then there are at most d/k primes of length k dividing D and the number of primes of length k is O $(2^k/k)$. Thus the probability that p divides D is at most $\mathrm{O}(d/2^k)$. Since D $\leq t!M^t$ the lemma follows. ■

Corollary 1: Let $w^{(1)}, \dots, w^{(t)}$ be t incoming vectors that are linearly independent in $\mathbb{Z}$. The probability that they are dependent modulo a random prime of size k is $\mathrm{O}(Lm \log mq/2^k)$.

Proof: If the $w^{(i)}$ vectors are valid then they are in the span $w^{(1)}, \dots, w^{(m)}$ and therefore the u-parts of these vectors, $u^{(1)}, \dots, u^{(t)}$, are also independent in $\mathbb{Z}$. The probability that these vectors are linearly dependent modulo a random prime of length k follows from the lemma by noting that $t \leq m$ and the maximal coordinate in any vector $u^{(i)}$ is at most $M = (mq)^L$. ■

If we consider m and L to be at most 128 and $q \leq 2^{16}$, and we set the length of the random prime p to be of *exact* length 32 (i.e., its most significant bit is set to 1) then the probability that the test mod p answers *dependent* on a set of vectors that is linear independent over $\mathbb{Z}$ is (if we disregard the constants in the O($\cdot$) notation) $2^{19}/2^{32} < 1$ / 8000. Actually, according to [25] the number of primes p such that $2^{31} < p < 2^{32}$ is 98,182,656. Thus, for a given determinant bound d as above, the probability that the linear independence fails modulo a 32-bit prime is at most $d/2^{31.5}$ (exactly d divided by 32 $\times$ 98182656 $=$ 3,141,844,992 $= 2^{31.5489649}$). With these adjustments, for the bounds in the above example the probability still gets a probability less than $1/8000$.

References

[1] MIRACL library. http://www.shamus.ie/.

[2] S. Agrawal and D. Boneh. Homomorphic macs: Mac-based integrity for network coding. In *ACNS*, pages 292–305, 2009.

[3] R. Ahlswede, N. Cai, S. yen Robert Li, R. W. Yeung, S. Member, and S. Member. Network information flow. *IEEE Transactions on Information Theory*, 46:1204–1216, 2000.

[4] P. S. L. M. Barreto and M. Naehrig. Pairing-friendly elliptic curves of prime order. In *Selected Areas in Cryptography*, pages 319–331, 2005.

[5] D. Boneh, D. M. Freeman, J. Katz, and B. Waters. Signing a linear subspace: Signature schemes for network coding. In *Public Key Cryptography*, pages 68–87, 2009.

[6] S. Chachulski, S. Chachulski, M. Jennings, M. Jennings, S. Katti, D. Katabi, and D. Katabi. Trading structure for randomness in wireless opportunistic routing. In *Proc. of ACM SIGCOMM*, 2007.

[7] D. Charles, K. Jain, and K. Lautner. Signatures for network coding. In *40th Annual Conference on Information Sciences and Systems (CISS 06)*, 2006.

[8] P. A. Chou, Y. Wu, and K. Jain. Practical Network Coding. In *Allerton'03*, Monticello, IL, Oct. 2005.

[9] J. Dong, R. Curtmola, and C. Nita-Rotaru. Practical defenses against pollution attacks in intra-flow network coding for wireless mesh networks. In *WISEC*, pages 111–122, 2009.

[10] W. Freeman and E. Miller. An experimental analysis of cryptographic overhead in performance-critical systems. In *MASCOTS '99: Proceedings of the 7th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, page 348, Washington, DC, USA, 1999. IEEE Computer Society.

[11] P. Ganesan, R. Venugopalan, P. Peddabachagari, A. Dean, F. Mueller, and M. Sichitiu. Analyzing and modeling encryption overhead for sensor network nodes. In *WSNA '03: Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications*, pages 151–159, New York, NY, USA, 2003. ACM.

[12] R. Gennaro, J. Katz, H. Krawczyk, and T. Rabin. Secure network coding over the integers. In *Public Key Cryptography*, pages 142–160, 2010.

[13] C. Gkantsidis and P. Rodriguez. Network coding for large scale content distribution. In *IEEE Infocom 2005*, Miami, Florida.

[14] C. Gkantsidis and P. Rodriguez. Cooperative security for network coding file distribution. In *INFOCOM*, 2006.

[15] The GNU Multiple Precision Arithmetic Library. http://gmplib.org/.

[16] R. Johnson, D. Molnar, D. X. Song, and D. Wagner. Homomorphic signature schemes. In *CT-RSA '02: Proceedings of the The Cryptographer's Track at the RSA Conference on Topics in Cryptology*, pages 244–262, London, UK, 2002. Springer-Verlag.

[17] M. N. Krohn, M. J. Freedman, and D. Mazieres. On-the-fly verification of rateless erasure codes for efficient content distribution. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 226–240, 2004.

[18] A. Le and A. Markopoulou. Locating byzantine attackers in intra-session network coding using spacemac. In *NETCOD*, 2010.

[19] U. Lee, J.-S. Park, J. Yeh, G. Pau, and M. Gerla. CodeTorrent: Content Distribution using Network Coding in VANETs. In *MobiShare'06*, Los Angeles, CA, Sep. 2006.

[20] J. Liu, D. Goeckel, and D. Towsley. Bounds on the gain of network coding and broadcasting in wireless networks. In *INFOCOM*, pages 1658–1666, 2007.

[21] W. Liu, R. Luo, and H. Yang. Cryptography overhead evaluation and analysis for wireless sensor networks. In *CMC '09: Proceedings of the 2009 WRI International Conference on Communications and Mobile Computing*, pages 496–501, Washington, DC, USA, 2009. IEEE Computer Society.

[22] D. J. C. MacKay. Fountain codes. *IEE Proc.-Commun.*, pages 1062 – 1068, 2005.

[23] R. W. Y. Ning Cai. Network coding and error correction. In *Information Theory Workshop*, pages 119–122. IEEE Computer Society, 2002.

[24] R. W. Y. Ning Cai. Network Error Correction, II: Lower Bounds. Commun. pages 37–54. Communications Infomation and Systems, 2006.

[25] Number of primes p such that $2^n < p <= 2^(n+1)$. http://oeis.org/A036378.

[26] S. Y. Oh and M. Gerla. Network coding multicast performance when some nodes do not code. In *PROC. OF The Seventh International Conference on Wireless On-demand Network Systems and Services*, 2010.

[27] J.-S. Park, M. Gerla, D. S. Lun, and M. M. Y. Yi. Codecast: A network-coding-based ad hoc multicast protocol. *IEEE WIRELESS COMMUNICATIONS*, 13(5):76–81, 2006.

[28] Scalable Networks. http://www.scalable-networks.com.

[29] M. Scott, August 2010. Private communication.

[30] A. Shokrollahi. Raptor codes. *IEEE Trans. Inform. Theory*, pages 2551 – 2567, 2006.

[31] F. Zhao, T. Kalker, M. Medard, and K. J. Han. Signatures for content distribution with network coding. In *Proc. of International Symposium on Information Theory (ISIT)*, 2007.