Quantitative Evaluation of Secure Network Coding using Homomorphic Signature/Hashing

Seung-Hoon Lee, Mario Gerla
University of California, Los Angeles
{shlee, gerla}@cs.ucla.edu

Hugo Krawczyk, Kang-Won Lee
IBM T.J. Watson Research Center
{hugokraw, kangwon}@us.ibm.com

Elizabeth A. Quaglia
Royal Holloway, University of London
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 protect network coding from 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 built on the field of integers which has exhibited 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 Homomorphic Network Coding under various network conditions and have compared it with other Secure Network Coding approaches. We conclude the paper with a discussion of secure coding feasibility and cost for different application scenarios.**

I. Introduction

Network coding [3], [18] has gained significant attention by improving network throughput and reliability. Basically, network coding offers a robust way to recover the original information over unreliable wireless links (e.g., jamming, interference, etc.). For example, let us consider a network with a source node that intends to share a file with multiple receivers. Conventionally, the file is divided into multiple blocks and the chunks of the file 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, the receivers are still able to recover the original data from the remaining coded blocks.

Despite the fact that network coding is designed to improve network performance and reliability, it is nevertheless vulnerable to *pollution attacks*. A malicious node can inject bogus coded blocks in the network; even a single invalid block can subvert the reconstruction of the original generation and severely affect performance. In order to prevent the attacks, several *secure network coding* mechanisms have been proposed [5], [7], [11], [15], [16], [29]. Basically, a secure network coding scheme provides intermediate nodes with a method to verify the validity of coded vectors. We can classify secure network coding approaches into two categories, *homomorphic signature scheme* [5], [7], [15] and *homomorphic hash functions* [16], [29]. Both schemes rely on the homomorphic property that the signature (or hashing) of a random linear combination of blocks is equivalent to the homomorphic combination of the signatures (or hash values). For example, in homomorphic signature schemes, when a node (either a source or intermediate node) generates a coded block, it also creates a signature of the block by using the same random coefficients applied to network coding. Note that the signature cannot be forged because the private key is known only to the source. Upon receiving a coded block, a node verifies its signature using the public key; if the signature does not pass the test, the coded block is discarded. This mechanism allows neighbor nodes to *immediately* detect pollution attacks; without network coding security, the attack would be discovered at the end point when a receiver tries to decode the original information. Thus, we preserve network coding performance by instantly detecting attackers and dropping infected packets.

While homomorphic cryptography can protect network coding from the 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

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.

operations) at each intermediate node with high computational overhead. Nodes always need to compute a new signature (or a hash value) after creating a new coded block. They also must verify the signature (or the hash value) whenever receiving a coded block. Thus, the computations can significantly affect to application performance, especially when nodes are resource constrained (e.g., processing power, memory space, battery capacity, etc.) [9], [10], [19]. In real network environments, there exit heterogeneous types of nodes with different computational capabilities. (i.e., typically smartphones have more resource constraint than conventional laptops.) In addition, various information flows require different levels of security enforcement. For example real time streams have a life expectancy of seconds and can be adequately protected with 64bits keys whereas U.S census data must lasts more than 100 years and requires much larger keys to preserve confidentiality of the information [26]).

Thus, in this paper, we investigate the processing overhead of homomorphic based secure network coding in real network environments considering heterogeneous node types and information flows and evaluate the feasibility of the implementation. We first implement in Linux the RSA-based homomorphic signature scheme and the homomorphic hashing scheme from [11]. This choice is justified as these schemes offer the lowest theoretical computation complexity when compared to other existing (public-key) secure network coding schemes. A detailed discussion and performance analysis is provided in Section V, where we show the advantages of the schemes we implemented, especially in terms of computational costs. 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 the QualNet network simulator [25].

The remainder of this paper is organized as follows. We discuss related work in Section II, and overview network coding and homomorphic signature and hashing schemes [11] in Section III. 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], [18] but also at the more practical level of protocol operations required under various scenarios [6], [12], [17], [23], [28]. 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 [17], [23] 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 methods to protect network coding have been proposed. Unfortunately, NC (Network Coding) blocks cannot be signed at the source with a conventional signature method and checked along the path because with NC the conventional signature is not preserved through linear combinations. A special signature method based on homomorphic functions is required. The homomorphic operations however tax the processing at nodes and are generally viewed as non feasible for field operations on mobile devices. To overcome this obstacle, several very creative schemes have been proposed in the past. Inspired by TESLA [24], Dong et al. propose DART [8] which is a time-based authentication strategy against pollution attacks that does not involve complex cryptographic operations. DART however requires nodes to verify a public key (called checksum) for each generation. Frequent broadcasting of the checksums increases network overhead. In addition, ,intermittent network, a common occurrence in tactical and emergency applications, prevents the delivery of the checksums and delays packet verifications. [20], [21] have proposed another secure network coding framework and have suggested theoretical bounds for network error correction.

Secure network coding schemes based on homomorphic signatures [5], [7], [11], [15] and homomorphic hash functions [11], [16], [29] have also been investigated. These schemes compute a signature (or hash value) that works even after recoding. Namely, the signature is regenerated every time the block is network encoded, and provides a method to verify the validity of any incoming blocks. Yu et al. design a homomorphic signature scheme which allows nodes to verify the signature of coded blocks without access to private keys [30]. Zhao et al. introduce a coded block authentication method by computing orthogonal vectors of each coded block [29]. Gennaro et al. propose an RSA based homomorphic signature scheme and a homomorphic hashing scheme for network coding over integers [11]. 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 [9], [10], [19]. In [9], 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

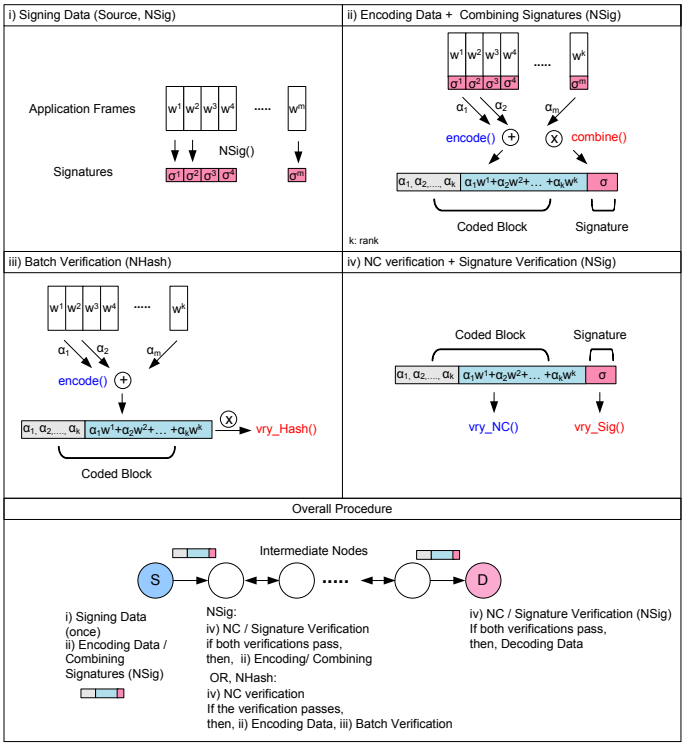


Fig. 1. Secure Network Coding Operations

of nodes in wireless sensor networks [19]. 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 [10]. 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 the homomorphic based secure network coding in portable platforms such as PCs and smart phones.

III. Secure Network Coding

In this section, we first review the *random linear* network coding scheme, then briefly describe the secure network coding scheme presented in [11] that is based on homomorphic signature and hashing over the integers.

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 $\mathbf{e}_\ell$ which is called the *encoding vector*. 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 $\mathbf{F}$.

B. An RSA based Network Coding Signature Scheme

In the RSA-based homomorphic scheme [11] 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 + 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 used 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 ($\approx$ 256). 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 e is the public RSA exponent and is chosen as a prime larger than mB^*. Let $\mathbb{G}$ be a cyclic group of order q (one of prime numbers composing N) , and let the public key contain a description of $\mathbb{G}$ along with random generators, $g_1, g_2, \ldots, g_n \in \mathbb{G}$. 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**, $\mathsf{NSig}(w)$**:**

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 $\mathsf{NSig}(w)$ is attached with w.

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

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

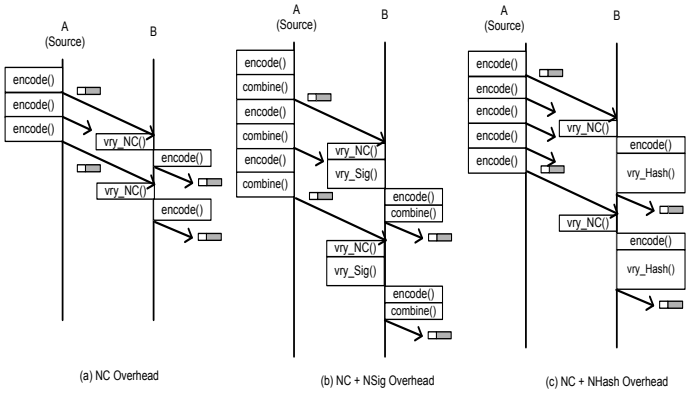


Fig. 2. Communication Models

retrieves the public key and verifies the block. w is accepted ($\mathsf{vry\_Sig}(w, \sigma, \mathbf{S}, \mathsf{fid})$ = 1) if and only if

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

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

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:

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

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

C. Homomorphic Hashing

[11] proposes another way of collusion-resistant scheme by using hash functions. 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 [11] 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

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

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.

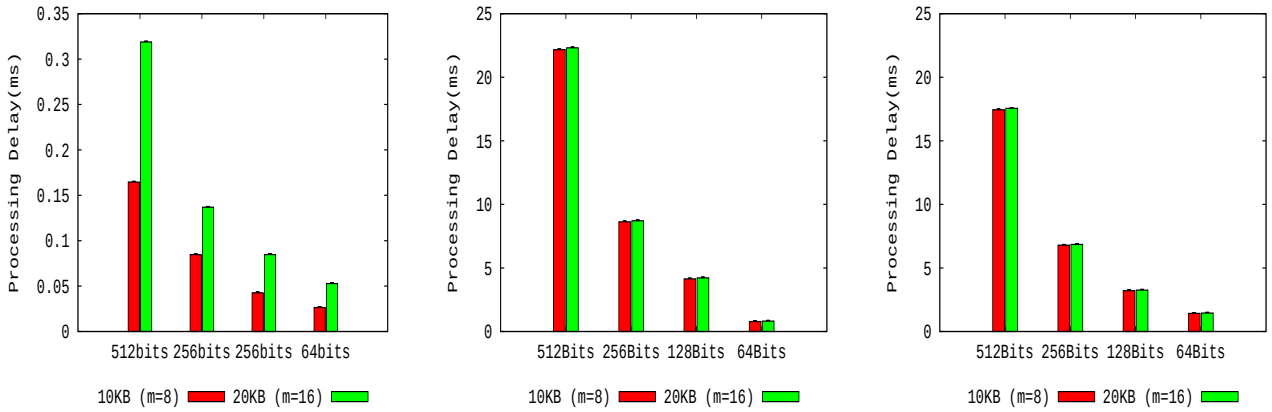
•**Homomorphic Hashing:** Unlike the homomorphic signature scheme, the hashing method verifies hash values after generating a new coded block. After A finishes *encode()*, it verifies the hash value by invoking *vry_Hash()*. If it fails, it means that some of blocks were falsified, and the generated 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.

V. Evaluation

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

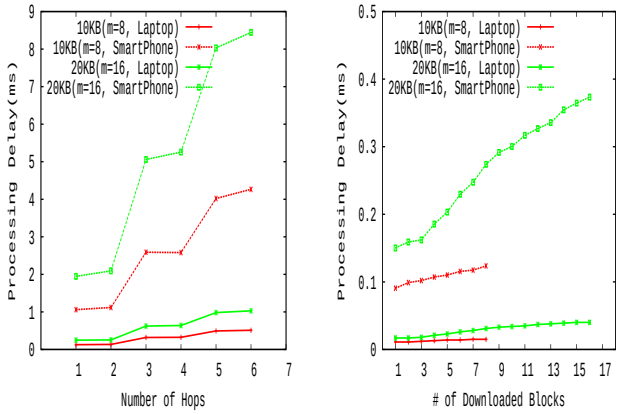
A. Processing Delay Experiments

In this section we report processing delays first for the basic random linear network coding operations described in Section III-A, and then for 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 [14]. Based on the implementation, we measure the processing delay of each operation on Linux platform with an Intel Core 2 Duo T9600 processor (2.80GHz, 6MB Cache) and Android platform with an ARM 11 processor



(a) Signature Generation Delay (b) Signature Verification Delay (c) Hash Verification Delay

Fig. 4. Processing Delay: Secure Network Coding



(a) Encoding Delay (b) NC Verification Delay

Fig. 3. Processing Delay: Network Coding

(528MHz). We use generation sizes [1], 10KB/20KB with the number of blocks 8/16 respectively, and we set the size of RSA modulus (N) as 64, 128, 256, and 512 bits.

•**Network Coding:** Network coding is mainly composed 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 network coding operates over the Integers, coefficients and sums grows each hops and it consequences of increasing computational delay on *encode()* with more hops. *encode()* delay increases from 0.12ms at source to 0.51ms at an intermediate node (6th hop).

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

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, nodes verify linear dependency of the block with the other blocks they previously received by running *vry_NC()*. It turns out 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 more coded blocks downloaded. We maintain a $m \times m$ matrix from coefficients of downloaded coded blocks and perform Gaussian elimination over the Integers. The detailed explanation of Gaussian elimination over the Integers is described in the Appendix. Thus, as a node has more blocks downloaded, Gaussian elimination requires more processing time.

In Fig. 3, we also present 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, delays of encoding and network coding verification operations on smartphone are around 9.2 times slower 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, say cellular phones, simply forward packets intact; yet, they detect and eliminate duplicates via linear dependency checks [22].

•**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 of 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 only for the Linux Platform. We measure the delay of *combine()* at a source node combining m signatures. Fig. 3(a) 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 where the size of RSA modulus is 512bits.

Upon receiving a coded block with a signature, intermediate nodes or destination nodes validate the signature by running *vry_Sig()*. 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 Fig. 4(b), and they are mainly determined by the symbol size and number of blocks in a generation.

The homomorphic hashing scheme does a batch verification by checking *vry_Hash()*. 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.

Considering the different security requirements for different information flows, we vary the length of the RSA modulus key from 64bits to 512bits in Fig 4. Processing delays of secure network coding operations decrease faster than linearly as RSA modulus size decreases. This suggests that there is a tradeoff between confidentiality of information and throughput performance in the case of streaming applications with high data rates and short lived sensitivity (e.g., situation awareness broadcasts).

As amply predicted in the published literature, we note that homomorphic processing delays using integer fields are much higher than the delays of conventional, Galois field based network coding. This is due to two reasons. First, non secure network coding operations mainly consist of pairs of multiplications and additions, whereas the homomorphic operations require a significant number of exponentiations and multiplications. Secondly, the integer field is not a closed field like the Galois field, thus the length of the operands increases linearly with the number of hops and so does the processing overhead. This is the price one must pay to secure the network coding operations. In the next section we discuss the impact of this increased processing cost on network performance.

B. Simulation

1) Simulation Setting: We implemented random linear network coding with the homomorphic signature and hashing schemes in QualNet simulator [25]. 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, the RSA modulus (N) value (i.e., key) is varied from 64bits to 512bits. The topology (see Fig. 5), has 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 = 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. The main performance metric under study 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. For each configuration, we report the value averaged over 100 runs with 95% confidence interval.

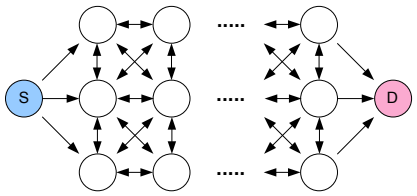
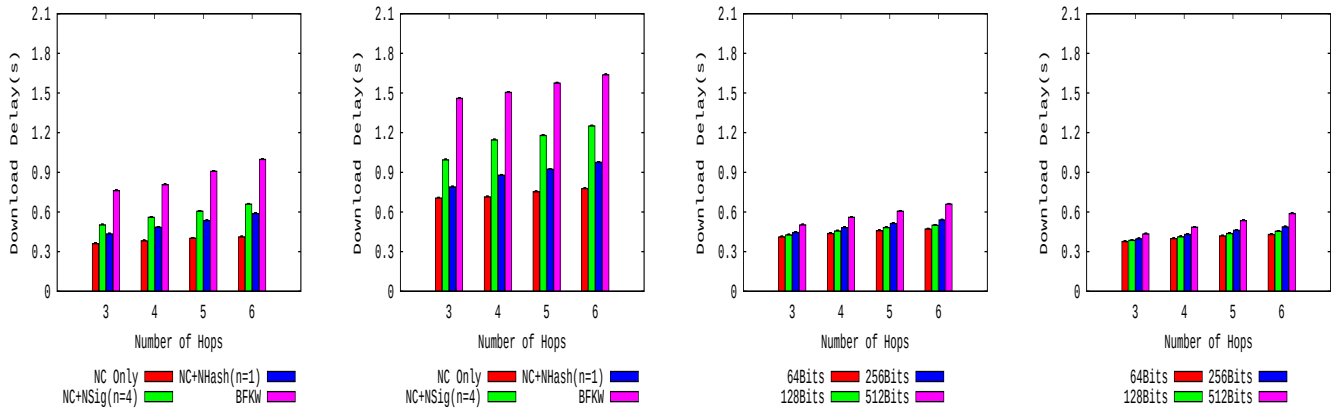


Fig. 5. Network Topology

2) Simulation Results: We compare the simulation results of the homomorphic signature and hashing 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 alternate homomorphic signature scheme, named 'BFKW', that was proposed in [5]. The detailed performance comparison of our scheme with 'BFKW' is explained in Section VI.

In Fig. 6(a) and Fig. 6(b), we compare the download delay for each scheme, for different hops from source to destination. 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 batch 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. 6(a), the generation size is set to 10KB and has 8 blocks, and; in Fig. 6(b), 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



(a) Download Delay (10KB, m=8, 512bits)

(b) Download Delay (20KB, m=16, 512bits)

(c) Download Delay with NSig (10KB, m=8)

(d) Download Delay with NHash (10KB, m=8)

Fig. 6. Impact of Hop Counts & RSA Modulus Sizes

blocks with a generation of size 20KB, the delay in Fig. 6(b) is higher.

In Fig. 6(c) and Fig. 6(b), we change the lengths of RSA modulus on both *NSig* and *NHash*. As discussed in Section V-A, processing delays of secure network coding operations mainly depend on the size of RSA modulus. The delay of *NSig* in Fig. 6(c) increases from 0.71 second to 0.80 second with RSA modulus size 64bits and 512bits respectively.

From the analysis of the results, we observe that Homomorphic Security causes a rather minor increase in stream delay performance over plain NC in the chosen experimental scenario. This is because the stream in the grid topology finds its bottleneck in the 2Mbps link speed (which introduces a transmission delay of 10,000/2Mbps = 5ms on each link, and thus a delay of 30ms on a six hop path) rather than in Linux PC processing delay (which is 20 ms in the 512 bit key size case). This result is important from a practical standpoint as it shows that Secure Network Coding can be supported in a representative battlefield scenario, dispelling claims to the contrary [8]. In order to expose the Secure Network code processing overhead, future experiments will consider higher link speeds (say, in unicast mode) and less performing platforms such as Smart Phones.

VI. Comparing performance of alternative schemes

The main goal of this paper is to assess the practical feasibility of cryptographic network coding techniques based on public-key cryptography. These techniques enable a full solution to the pollution problem in network coding by allowing intermediate and destination nodes to single out any contaminated packets without having to place any trust on any party other than the information source. At the same time, the computational cost of public-key techniques is significant and hence the practical applicability of this approach needs to be assessed. Before we expand more on available public-key based solutions, we note that pollution solutions based on symmetric-key techniques have been suggested. However, while these solutions are computationally more efficient they suffer from significant limitations. Specifically, the "secure random checksums" technique from [13] 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 [8] is of limited applicability as it requires strong assumptions on network connectivity and synchronization under adversarial attack.

The public-key based solutions can be classified in two groups: Those based on homomorphic hashing and those based on homomorphic signatures. 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.

In this paper we assess the practicality of both approaches by implementing the homomorphic hashing and homomorphic signature schemes proposed in [11]. 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.

In what follows we show how the alternative solutions, namely, the homomorphic signature scheme of [5] and the

homomorphic hashing schemes originating from [16], are computationally more expensive than the RSA-based solutions from [11]. (See our description in Sections III-B and III-C.)

As said, 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 [11] can be applied to the original schemes from [5] and [16] 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 [11] 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 [11] 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], [27] 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 [16] 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 [16] 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 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 [11]) 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

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

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 [11], 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* malicious attacks, several schemes have been proposed recently based on homomorphic properties of cryptosystems. In this paper, we implemented an efficient secure network coding scheme based on the field of integers. We have integrated 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 and Raptor codes) 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 $u_1, \ldots, u_\mathrm{t}$ be linearly independent vectors in $\mathbb{Z}^m$. The probability that $u_1, \ldots, u_\mathrm{t}$ are linearly dependent mod p is at most $\mathrm{O}(\log(t!M^t))/2_k$ where M is the largest coordinate in $u_1, \ldots, u_\mathrm{t}$.

Proof: Let p be a prime of length k such that $u_1, \ldots, u_\mathrm{t}$ are linearly independent over $\mathbb{Z}$ but dependent mod p. Consider the t $\times$ m matrix U whose rows are $u_1, \ldots, u_\mathrm{t}$. 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 $w^{(1)}, \ldots, w^{(t)}$ 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 $w^{(1)}, \ldots, w^{(m)}$ and therefore the u-parts of these vectors, $u^{(1)}, \ldots, u^{(t)}$, 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 $\mathrm{O}(\cdot)$ notation) $2^{19}/2^{32} < 1$ / 8000. Actually, according to http://oeis.org/classic/A036378 the number of primes p such that $2^{31} < p < 2^{32}$ is 98,182,656 (see explicit table under http://oeis.org/classic/table?a = 36378&fmt = 4). 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 one 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. Freeman, J. Katz, and B. Waters. Signing a linear subspace: Signature schemes for network coding, 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, 2006.

[8] 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.

[9] 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.

[10] 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.

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

[12] C. Gkantsidis and P. Rodriguez. Network coding for large scale content distribution. 2005.

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

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

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

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

[21] R. W. Y. Ning Cai. Network error correction, ii: Lower bounds. commun. pages 37–54. Communications Infomation and Systems, 2006.

[22] 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.

[23] 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.

[24] A. Perrig, R. Canetti, J. D. Tygar, and D. Song. The tesla broadcast authentication protocol, 2002.

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

[26] B. Schneier. John wiley & sons, 1996.

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

[28] P. C. Yunnan, P. A. Chou, Y. Wu, and K. Jain. Practical network coding, 2003.

[29] 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.

[30] B. R. Zhen Yu, Yawen Wei and Y. Guan. An efficient signature-based scheme for securing network coding against pollution attacks. In *Proceedings of INFOCOM*, pages 1409–1417, Phoenix, AZ, 2008. IEEE Computer Society.