Efficient Signature Scheme for Network Coding

Ely Porat* and Erez Waisbard*
* Department of Computer Science, Bar-Ilan University, Ramat-Gan, 52900, Israel
Email: {porately,waisbard}@cs.biu.ac.il

***Abstract*—Network coding helps maximize the network throughput. However, such schemes are also vulnerable to pollution attacks in which malicious forwarders inject polluted messages into the system. Traditional cryptographic solution, such as digital signatures, are not suited for network coding, in which nodes do not forward the original packets, but rather linear combinations of the data they received. We describe secure scheme that uses batch techniques and selective verification to efficiently verify the integrity of the received packets. We show that for real peer-to-peer networks, our scheme is much more efficient than previously suggested schemes.**

***Index Terms*—Network coding; Digital signatures; Homomorphic signatures; Group testing; Peer-to-peer; File sharing**

I. Introduction

The volume of content that is distributed over the internet is growing every day. The simple client-server based approach finds it difficult to cope with the growing number of clients and volume of data and there is a growing usage of peer-to-peer technologies.

A peer-to-peer network uses distributed architecture instead of relying on a single server. The benefit of this approach is in the splitting of the workload between many different servers instead of overloading a single server. BitTorrent [5] is an example for a popular peer-to-peer network that is used to distribute large files over the Internet. It works by splitting the large file into smaller blocks and sending these blocks to peers. Any peer that downloaded a block also becomes a server to this block and so the number of available sources increases as more peers download it. However, as efficient as BitTorrent may be, it may happen that the scheduling scheme would result in having some blocks available only from a few peers with slow connection and as a result the completion of downloading of the entire file is delayed for many users. Using network coding as an efficient way to cope with this problem was suggested in [2], [8].

Network coding [3] was introduced as an efficient alternative to traditional routing that maximizes network throughput [10], [13], [14]. Network coding improves the spreading of different blocks in a peer-to-peer network by sending a random linear combination of the blocks instead of sending a single block. In such a scheme each peer continues to act as a server for blocks it has obtained and sends out random linear combinations of the blocks it has received. After receiving enough packets a peer would be able to reconstruct the file from the system of linear equations. Note that even before a peer is able to extract the different blocks, it can already participate in the distributed effort to spread the content by creating a new random linear combination of the data that it already received. Using this method every peer simultaneously sends and receives partial information about many block, thus solving the problem of rare blocks.

A. Securing Network Coding

One can see that network coding improves the network throughput and works well as long as everyone follows the protocol. Unfortunately, this is not always the case in real systems. Real systems need to cope with malicious users who try to pollute the network. In traditional content distribution schemes, this problem is easily solved by verifying the integrity of the packets using cryptographic tools such as hash functions and digital signatures. This is also the case in peer-to-peer networks such as BitTorrent where each block of data has a hash value stored in the *.torrent* file. Each packet that is received by the peer is verified by computing its hash value and comparing it to the value that appears in the *.torrent* file. The security of a cryptographic hash function ensures that it is infeasible to find two different blocks that hash to the same value. The integrity check takes place for each block separably. This allows early detection of corruption without having to wait for the entire file to download and prevents the further spread of corrupted packets.

Securing network coding schemes is more difficult than securing traditional peer-to-peer schemes. We first note that the problem of polluting the network is more acute for network coding schemes as a single corrupted block can corrupt the entire file and prevent its reconstruction. Furthermore, such a pollution quickly propagates throughout the network. Thus, it is crucial to be able to verify the integrity of incoming packets. Alas, the effective traditional cryptographic techniques that secure peer-to-peer networks such as BitTorrent no longer work. Incoming packets in a network coding scheme are random linear combinations of the source blocks and can take many different values. Since one cannot tell in advance which combination would be used by the forwarding peers, it is impossible to compute the appropriate hash values in advance.

II. Previous Work

Some attempts have been made to solve this problem. These attempts largely focus on settings that are different than the one needed for large distribution file sharing over the internet.

Trying to solve the problem from an information theoretic angle, [6], [17] suggested a way to protect the network against a passive adversary. Clearly this does not provide the required protection from an adversary in our setting.

Others considered byzantine adversaries [10], [9], [11] and provide security against a known threshold, but since adversaries can freely join public peer-to-peer network, they can easily outnumber the byzantine bound.

A different approach is to use homomorphic MACs [1]. Loosely speaking, Homomorphic MACs allows nodes that do not know the secret key to create a valid tag that can be authenticated by the end recipient. This allows the end recipient to discard polluted packets, but does not prevent their distribution throughout the network. This solution also assumes the existence of a shared secret between a sender and a receiver, which is not applicable to many settings. In particular, most of the popular peer-to-peer networks work in a mode in which one user shares a copy of his files with many users which he does not know in advance.

Homomorphic signatures [18] allow intermediate nodes to create a signature on a linear combination of the incoming messages [18] or on any polynomial function of them [4]. Homomorphic signatures start by the source signing the different blocks. Each peer gets a linear combination of the blocks along with an homomorphic signature of it. The homomorphic property enables creating a new signature from the previously received signatures. These signatures are limited to signing only linear combinations of previously received packets, which is exactly what we need for network coding. This allows intermediate nodes to verify the integrity of the messages they receive and fits well with the adversarial model for peer-to-peer file sharing. However the known schemes are either inefficient [4] or known to be broken [18].

Another approach by [12] was to use the space orthogonal to the message space as a mean to verify that incoming messages are from a valid message space. This method is indeed more efficient, however, as was noted by [16] the need to securely distribute the NULL message space before one can efficiently use this scheme puts us in a chicken and egg situation.

A. Signing Orthogonal Vector

Zhao, et.al. [19] introduced an interesting scheme in which a source can 'sign' the message in a way that allows intermediate nodes to efficiently verify the integrity of the messages without requiring a prior shared secret. Their scheme works by splitting the message M into vector blocks $(v_1, \ldots, v_n)$ and looking at the vector space that they span $V = Sp(v_1, \ldots, v_n)$. Then they find a random vector u which is orthogonal to the message space and 'sign' it using a standard signature scheme. The signed vector is sent as part of the public information (the equivalent of the *.torrent* file in the BitTorrent). Using the linearity property u is guaranteed to be orthogonal to every linear combination of the message blocks. Upon receiving an incoming message, the receiving peer verifies that the incoming message is orthogonal to u as a proof for it being a valid combination of the original blocks. Note that there are many vectors that are orthogonal to u, but are not in V. Thus, it is important to be able to verify that a vector is indeed orthogonal to u, but without explicitly publishing u. Using the hardness assumption of the discrete logarithm problem, they publish a commitment to u that allows checking if a given vector is orthogonal to u without using u directly. We elaborate on the way it is done when we provide details of our construction in section III. The number of modular operations that are required in this scheme is proportional to the size of the vectors. If one considers a typical file sharing scenario over the internet, then a back of the envelope computation indicates that the verification time of an incoming packet is longer than the communication time to receive it. Our construction aims to improve the efficiency of the verification step by significantly reducing the number of modular operations per packet.

III. Our Construction

We present a method for boosting the performance of homomorphic signatures. In this paper we present our construction by applying it to a specific scheme that was presented in [19]. Our method significantly reduces the number of modular exponentiations required per packet by introducing two techniques: Selective verification and Batch verification.

We start with a large file F that we wish to share and divide it into n blocks, denoted $\bar{v}_1, \ldots, \bar{v}_n$. Each block is viewed as an element in the $m-$dimentional vector space $\mathbb{F}_p^m$, where p and q are large primes such that p is a divisor of $q-1$.

Each vector is then assigned with a prefix and these augmented vectors $v_1, \ldots, v_n$ are given by

$$v_i = (\underbrace{\overbrace{0, \ldots, 1}^{i}, \ldots, 0}_{n}, \underbrace{\bar{v}_i}_{m})$$

where the first n elements are zero except that the ith one is 1, and $\bar{v}_i \in \mathbb{F}_p^m$ is the i-th vector block from the file.

Let us denote the subspace that is spanned by these vectors as V. Namely:

$$V = Sp(v_1, \ldots, v_n)$$

A. Basic Flow

The source starts distributing the data by computing a random linear combinations of the augmented vectors and sending them to peers. After collecting enough of these vectors the receiving peer would have n linearly independent blocks and would be able to reconstruct the original file. Peers actively participate in the distribution of the file (even before they are able to reconstruct the original file) by sending new random linear combinations of the blocks they obtain.

In order to protect against pollution attacks, we use an orthogonal vector to V as an indication that the incoming blocks are indeed from V. Noting that the augmented vectors span a subspace V of $\mathbb{F}_p^{n+m}$, there exist many vectors which are orthogonal to V. Let us denote one of these vectors by $u = (u_1, \ldots, u_{m+n})$. As $u \perp V$, it is also orthogonal to any linear combination of the augmented vectors. Thus, checking that an incoming block is orthogonal to u would serve as an indication that it belongs to V. Naturally, the peers would need to know u before they can verify the authenticity of the incoming blocks. The vector u also needs to be authenticated and this can be done by having u digitally signed by s (using any standard digital signature scheme).

However, if an attacker knows u then he can easily find $v \notin V$ for which $< u, v >= 0$. Thus we hide u by utilizing the hardness assumption of the discrete logarithm problem. Instead of using u directly, s picks a large prime q, such that p is a divisor of $q-1$ and a generator g of a group of order p in $\mathbb{F}_q^*$. Then s picks a secret set of random elements in $\mathbb{F}_p^*$: $\{\alpha_i\}_{i=1,\ldots,m+n}$ and publishes $\{h_i = g^{\alpha_i}\}_{i=1,\ldots,m+n}$ after digitally signing it (again, using any standard digital signature scheme). Then s finds $u_1, \ldots, u_{m+n}$ orthogonal to all vectors in V and computes a vector $x = \frac{u_1}{\alpha_1}, \ldots, \frac{u_{n+m}}{\alpha_{n+m}}$, which he also signs.
When a node receives a vector w, it computes

$$d = \prod_{i=1}^{m+n} h_i^{x_i w_i}$$

and verifies that $d = 1$ and that x is properly signed[1].

B. Selective Verification

The first idea that we introduce is the *selective verification*. Loosely speaking, we are going to verify only part of the w_i, but still get a high level of assurance while checking a significantly smaller number of bits. In order to ensure that one cannot avoid this selective check by corrupting a small number of bits, we are using error correcting codes that expand the incoming messages in a way that any number of corrupted bits is reflected in many of the expanded bits.

The use of the error correcting code allows us to break the resulting expansion into ℓ blocks, each of length $(1+\epsilon)n$, and to do the verification in one or more of these blocks. The idea is that for the right values of ϵ, checking over one or more blocks is much faster than verifying the full length of w. We use an efficient construction of the code to ensure locality in the computation. Namely in order to verify a single block, we do not need to compute the full expansion of all the blocks. A key observation is that the probability of an attacker to pass this verification *using a corrupted vector* is small as the number of possible "legal" blocks is much smaller than ℓ.

We stress that although we use error correcting codes in the verification step, there is no change in the length of the messages that traverse the network. Thus, with the exception of the initial signature (that is part of the metadata) the communication complexity remains the same as in [19].

1) The Error correcting code: We start by defining the error correcting code that we use. While a known code like the Reed-Solomon code may be used with a small tweak, we have chosen to explicitly define a new code that allows to illustrate the main ideas of our scheme in the simplest way.

Our random code is a function $h : \mathbb{F}_p^{m+n} \mapsto \mathbb{F}_p^{(1+\epsilon)n\ell}$, defined as follows:

$$h(v) = (vA_1 | \ldots | vA_\ell)$$

[1]In practice the signature on x does not need to be verified each time a message is received. it suffices to check it once before the first messages is verified and then only verify that x and the signature remains the same

Where each of the A_is is random[2] $(m+n) \times (1+\epsilon)n$ matrix over $\mathbb{F}_p^*$.

2) The Algorithms: We now describe the three key components of our scheme. We start with a description of setting up the public parameters and in particular the verification vector. We then specify the sending algorithm and verification algorithm. We note that there is no change in the sending algorithm when compared to [19]. The change is in the verification step.

- ***Computing orthogonal vector:*** Instead of finding a vector u that is orthogonal to every v_i in the original file, we first extend the matrix using error correcting code (ECC).

  We look at the resulting matrix E and break it into blocks of size $(1+\epsilon)n$:

$$(I|M) \to (B_1|B_2|\cdots|B_\ell)$$

  Where each $B_i = (I|M)A_i$

  We then find vectors $u_1, \ldots, u_\ell$, where u_j is orthogonal to $Sp(B_j)$. For every vector u_j we also pick a secret set of random elements in $\mathbb{F}_p^*$: $\{\alpha_i^j\}_{i=1,\ldots,(1+\epsilon)n}$, compute $\{h_i^j = g^{\alpha_i^j}\}_{i=1,\ldots,(1+\epsilon)n}$, compute $\{x_1^j = \frac{u_i^j}{\alpha_i^j}\}_{i=1,\ldots,(1+\epsilon)n}$ and publish $\{x_i^j\}$ and $\{h_i^j\}$ along with their digital signatures.
- ***Sending a message:*** Sending a message is done as usual in network coding schemes by picking at random $a_i \in \mathbb{F}_p^*$ and sending $v = \sum_i a_i v_i$.
- ***Message verification:*** Upon receiving a vector w, the receiver picks a random $j \in \{1, \ldots, \ell\}$ as the index of the block to verify and applies the ECC locally (i.e. computes only wA_j).
  Let

$$wA_j = (z_1, \ldots, z_{(1+\epsilon)n})$$

  It then checks if $d = \prod_{i=1}^{(1+\epsilon)n} h_i^{x_i^j z_i}$ equals to 1.

The speed up we gain by this method is proportional to the size of the block that we check compared to the size of the original vector, i.e. roughly $\frac{(1+\epsilon)n}{m+n}$. In terms of security, it was proven in [19] that under the hardness assumption of solving the discrete logarithm problem, the scheme is secure if we were to check all blocks. We are going to check only portion of the blocks, but we are going to do so after applying the ECC. Intuitively, the ECC would ensure that even a few corrupted bits in the input vector would result in plenty of corrupted blocks. In section IV we show why even under a worse case scenario we can bound the error probability of our scheme with δ of our choice.

C. Batch Verification

Another performance enhancement comes from performing *batch verification*. The batch verification technique is independent of the selective verification technique. It can be used

[2]In order to make computations efficient we can pick A to be sparse

in conjunction with selective verification or independently (e.g. it can be applied directly to the scheme of [19]). Batch verification works by gathering a few incoming vectors before performing the verification. Only once enough vectors are gathered, we check all of them in single verification operation. The way we batch the vectors together is done as follows: Let $w_1, \ldots, w_k$ be the incoming vectors. We pick random $r_1, \ldots, r_k$ and compute $w' = r_1 w_1 + \ldots + r_k w_k$. Clearly, if w' is a linear combination of vectors that are orthogonal to u, then w' would also be orthogonal to u and it is enough to verify that w' is orthogonal to u. If some of the vectors are not orthogonal to u, then with high probability w' will not be orthogonal to u.

Since verification only occurs once every few packets, the incoming vectors $w_1, \ldots, w_k$ should be stored and marked as unverified until they pass verification. This is important in cases where verification fails and we wish to be able to identify which of the batched vectors are the corrupted ones (and perhaps also identify their sender).

As the modular operations are far more expensive than computing a linear combination of the incoming vectors, if we batch j vectors, our verification time would be j times faster.

This certainly helps in terms of saving on verification operations, but it also introduces a potential delay. If each node would need to wait for the arrival of t messages before it can forward them, then we are introducing a potential bottleneck.

In order to prevent this bottleneck we propose a randomized forwarding scheme that would forward packets even before they are checked, but with a lower probability. Before diving into the details, we use an example to illustrate our point. Assume that 10% of the incoming packets are polluted and assume that a forwarding probability of a non verified packet is 10%. This means that an unverified malicious packet would be forwarded with probability of 1%. The probability further forwarding corrupted packets quickly drops. In order to make this enhancement more robust, we suggest not to use a fixed probability, but rather start with a conservative forwarding policy that gives little to no chance of forwarding a packet from an unfamiliar source. Only after a period of time, in which no corrupted packets received from this source, the probability of forwarding unverified packets should increase.

D. Combining Selective and Batch Verification

The two techniques introduced above can work in conjunction. It can be readily seen that the overall performance boost is the product of the boost provided be each individual technique. We now make a note on how to combine the two techniques in a way that constantly increases the level of assurance. The naive way of combining both techniques is to batch k vectors each time, then perform a selective verification on their linear combination and later to repeat the process with a different group of incoming vectors. Since selective verification may err, a better approach is to add previously verified vectors to the batch. For example, if we batch k vectors at a time, then initially we would do a selective verification on the linear combination of the first k vectors. After we get an additional k vectors we would batch all of the $2k$ vectors using fresh $r_1, \ldots, r_{2k}$ before running selective verification on their linear combination. This way we double check the first k vectors (on different locations), reducing the chance that selective verification fails to spot corrupted vectors.

IV. Security

In this section we prove the security of our construction. It was proven in [19] that finding a vector $w \notin V$ for which $\prod_{i=1}^{m+n} h_i^{x_i w_i} = 1$ is as hard as the discrete log problem. What remains to show is that our selective verification does not allow the spreading of polluted file.

We take a group testing [15] approach to analyzing the security of our scheme. Using the ECC we defined in III-B we apply h to all the possible vectors in $Sp(V)$. Let us denote the number of vectors in $Sp(V)$ by k, i.e. $k = p^n$.

Given any input vector $y \notin \{x_1, x_2, \ldots, x_k\}$, we ask what is the probability that the $j-$th element in $h(y)$ (denoted $h(y)[j]$) equals the $j-$th element in one of the previous x_is. Summing over all the x_is (using the union bound) we get that

$$\sum_i Pr[h(y)[j] = h(x_i)[j]] = \sum_i Pr[x_i A_j = y A_j]$$

Since A_j is random and since $x_i \neq y$, the event in which $(x_i - y)A_j = \bar{0}$ occurs with probability

$$(\frac{1}{p})^{(1+\epsilon)n} = (\frac{1}{p^n})^{1+\epsilon} = \frac{1}{k^{1+\epsilon}}$$

Thus

$$\sum_i Pr[x_i A_j = y A_j] = k \frac{1}{k^{1+\epsilon}} = \frac{1}{k^\epsilon}$$

The probability that $\delta\ell$ elements in $h(y)$ appear in $h(x_i)$ is bounded by $\binom{\ell}{\delta\ell}\,(\frac{1}{k^\epsilon})^{\delta\ell}$

By taking the union bound over all possible values of y (which is bounded by p^{m+n}) we get that the probability for existence of a "bad" y is bounded by $p^{m+n}\binom{\ell}{\delta\ell}(\frac{1}{k^\epsilon})^{\delta\ell}$

In order to give a sense of security for practical parameters, lets assume that we want to bound[3] this probability by 2^{-40}. In this case we get that

$$p^{m+n}\binom{\ell}{\delta\ell}\left(\frac{1}{k^\epsilon}\right)^{\delta\ell} < 2^{-40}$$

i.e.

$$p^{m+n}\left(\frac{e}{\delta k^\epsilon}\right)^{\delta\ell} < 2^{-40}$$

Looking at the discrete log of the above equation we get

$$m + n + \delta\ell(log_p e - \log_p \delta - \log_p k^\epsilon) < -40$$

and since $k^\epsilon = (p^n)^\epsilon$ we get

$$m + n + \delta\ell(log_p e - \log_p \delta) - \epsilon n) < -40$$

[3]We note that the error probability is determined by the file that we protect and the ECC that we use and not by the adversary's strategy.

Since $log_p\delta$ and log_pe are constants smaller than 1 (assuming a typical p to be about 2^{1024}), we get that

$$m+n+\delta(-\epsilon n) < -41$$

Therefore, in order to have an error smaller than δ we would need to use $\epsilon > \frac{41+m+n}{\delta\ell n}$.

We now consider the size of the orthogonal vector that achieves the above security. If we take for example $\delta = \frac{1}{10}$ and we wish to use blocks of size $2n$ then in order to get the security assurance for $\epsilon = 1$ we would require our orthogonal vector to be of size $21(m+n)$. If we allow ourselves bigger blocks of size $3n$, then we take $\epsilon = 2$, then we can see that it is sufficient to stretch our orthogonal vector to the length of $16(m+n)$.

The different tradeoffs between ϵ and the stretch can be viewed in the graph below.

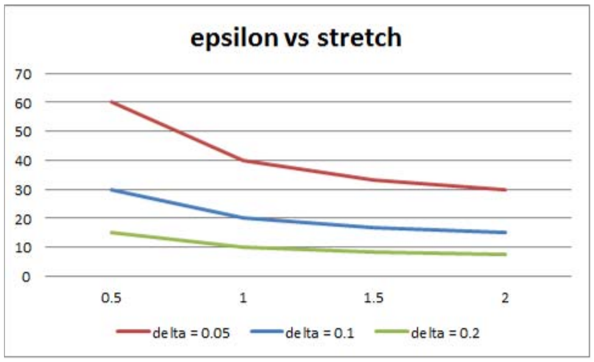

Fig. 1. epsilon vs stretch

A. Further security considerations

It was mentioned both in [19] and in [16] that the public key cannot be used for too many files as it would allow to collect enough equations to compute u. Thus a different public key needs to be associated with every new file. Since this information can be part of a .torrent file the only affect is that it adds a little overhead to the original data that is being transferred.

V. Performance

We now analyze the performance gains of our scheme and compare it to [19]. For this analysis we use common parameters from the Bittorrent peer-to-peer network.

The file size that we are going to share is 1GB. We are assuming a modulus q of 1024 bits[4] and we get that the packet size is $m = 4MB$ (i.e. 4096*8 values from $\mathbb{F}_p^*$ and there are $n = 256$ vectors in our matrix).

In the original scheme we had $m+n$ modular exponentiations, which are at least $logq * (m+n) = 1024(m+n) = 33816576$ modular multiplications.

[4]We note that p and q are of the same order of magnitude

In our method, let us take $\epsilon = 1$. We only have $(1+\epsilon)n$ modular exponentiations, i.e. $2 \cdot 256 = 512$ modular exponentiations, which are at least $512 * 1024 = 524288$ modular multiplications. In addition we have our ECC, which takes $(m+n)(1+\epsilon)n$ modular multiplications assuming the expansion matrix is full. However, if we take a sparse matrix with only 10 elements in each row[5], then we get that there are only $10(m+n) = 330240$ modular multiplication. So the total number of modular multiplication in our scheme is about 854528, which makes our scheme about 40 times faster than [19] with only using the selective check method. Further performance gain can be achieved by using batch verification. If we batch in groups of five, then we would get that our scheme is about $5 * 40 = 200$ faster than [19].

References

[1] S. Agrawal and D. Boneh. Homomorphic MACs: MAC-Based Integrity for Network Coding. In *ACNS 2009*. pages 292–305, Springer Verlag.

[2] S. Acedanski, S. Deb, M. Medard, and R. Koetter, How good is random linear coding based distributed network storage?, in Proc. 1st Workshop on Network Coding, Theory, and Applications (Netcod05), Riva delGarda, Italy, April 2005.

[3] R. Ahlswede, N. Cai, S. Li, and R. Yeung. In *Network information flow. IEEE Transactions on Information Theory*, 46(4). pages 1204-1216, 2000.

[4] D. Boneh, D. M Freeman. Homomorphic Signatures for Polynomial Functions. In *EUROCRYPT 2011*. pages 149–168

[5] BitTorrent file sharing protocol. http://www.BitTorrent.com

[6] N. Cai and R. Yeung. Network coding and error correction. In *Proc. 2002 IEEE Information Theory Workshop 2002*. pages 119-122

[7] R. Gennaro, J. Katz, H. Krawczyk, T. Rabin. Secure Network Coding over the Integers. In *Public Key Cryptography 2010*. pages 142–160

[8] C. Gkantsidis and P. Rodriguez, Network coding for large scale content distribution, in Proc. IEEE INFOCOM05, Miami, FL, March 2005.

[9] T. Ho, B. Leong, R. Koetter, and M. Medard. Byzantine Modification Detection in Multicast Networks using Randomized Network Coding. In IEEE Proc. ISIT 2004.

[10] T. Ho, M. Medard, M. Effros, and D. Karger, The benefits of coding over routing in a randomized setting, in Proc. IEEE Symposium on Information Theory (ISIT03), Kanagawa, Japan, July 2003.

[11] S. Jaggi, M. Langberg, S. Katti, T. Ho, D. Katabi, M. Medard Resilient network coding in the presence of Byzantine adversaries. In Proc. INFOCOM, pages 616624, 2007.

[12] E. Kehdi, B. Li, "Null Keys: Limiting Malicious Attacks Via Null Space Properties of Network Coding." in INFOCOM 2009 pages 1224–1232

[13] Z. Li and B. Li, Network coding: the case of multiple unicast sessions,in Proc. 42th Annual Allerton Conference on Communication, Control, and Computing, September-October, 2004.

[14] D. S. Lun, M. Medard, and R. Koetter, Network coding for efficient wireless unicast, in Proc. 2006 International Zurich Seminar on Communications (IZS06), Zurich, Switzerland, February 2006

[15] E. Porat, A. Rothschild, "Explicit Non-adaptive Combinatorial Group Testing Schemes". In ICALP (1) 2008 pages 748–759

[16] Y. Wang, Insecure "Provable Secure Network Coding". In *IACR Cryptology ePrint Archive 2009: 504*, 2009

[17] R. Yeung and N. Cai. Network Error Correction, In *Basic Concepts and Upper Bounds. Commun. Inf. Syst. 6(1), 2006* pages 19-35

[18] Z. Yu, T. Wei, B. Ramkumar, and Y. Guan. An Efficient Signaturebased Scheme for Securing Network Coding against Pollution Attacks. In *IEEE INFOCOM, 2008.*

[19] F. Zhao, T. Kalker, M. Medard, K. Han. Signatures for Content Distribution with Network Coding. In *Proc. 2007 IEEE ISIT*

[5]A good heuristic would be having a few elements in each row. In our computation we used 10 as a conservative value.