Computer Communications xxx (2010) xxx–xxx

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

Data obfuscation with network coding ☆

A. Hessler [a], T. Kakumaru [a], H. Perrey [b], D. Westhoff [b,*]

[a] NEC Europe Ltd., Kurfürsten-Anlage 36, 69115 Heidelberg, Germany
[b] Hamburg University of Applied Sciences (HAW), Department of Computer Science, Berliner Tor 7, 20099 Hamburg, Germany

ARTICLE INFO

Article history:
Received 17 June 2010
Received in revised form 21 September 2010
Accepted 10 November 2010
Available online xxxx

Keywords:
Data obfuscation
Fountain code
Privacy homomorphism
Content distribution

ABSTRACT

Network coding techniques such as fountain codes are a promising way to disseminate large bulks of data in a multicast manner over an unreliable medium. In this work we investigate how to conceal such an encoded data stream on its way to numerous receivers with a *minimum investment*. Compared to conventional 'encrypt – encode/decode – decrypt' approaches, our solution is preferable for two reasons: First, it causes less CPU investment for encryption and decryption proportional to the ratio of the payload length to the signaling data length. Second, besides obfuscating the payload data from an eavesdropper, we hide the coding information from an eavesdropper. We evaluate the approach with respect to its application to various data types like MPEG-2 video streams and Java bytecode.

1. Introduction

In network coding transmission, an intermediary node can combine different packets together in order to reduce the number of transmissions, and thus increase the overall throughput of the network. To disseminate bulks of data over an unreliable medium to numerous receivers, rateless erasure codes, aka fountain codes, are an efficient way to cope with various packet losses, and reduce the need of a feedback channel. In its simplest form, a packet is composed of a payload $\mathbf{x}$ of network encoded data, and the metadata $\mathbf{c}$, the coefficient vector. $\mathbf{c}$ carries the information which of the plaintext packets $\mathbf{p}_1,\ldots,\mathbf{p}_n$ contributed to form $\mathbf{x}$. We consider the case where the encoded packets are XOR combinations of several source packets, and the metadata are bit-vectors of size n-bit. The receiver can decode the received encoded packets by solving the linear equation system formed by the received $\mathbf{c}$ s.

In a multi-hop propagation scenario, a forwarding node may not only want to forward received encoded packets $(\mathbf{x}_i,\mathbf{c}_i)$ but also generate freshly encoded packets $(\mathbf{x}_f,\mathbf{c}_f)$ by combining a subset of the already received encoded packets. Therefore, the node is required to compute $\mathbf{x}_f = \oplus_{i=1}^{l}\mathbf{x}_i$ and $\mathbf{c}_f = \oplus_{i=1}^{l}\mathbf{c}_i$. In the state of the art, the coefficient vector $\mathbf{c}$ that describes how the different plaintext packets were combined remains in clear-text, i.e. every intermediate node knows, which source plaintext packets $\mathbf{p}_i$ contributed to the received encoded packet. For example, assuming that the data stream is subdivided into packets $\mathbf{p}_1,\ldots,\mathbf{p}_8$ and in case the $\mathbf{c}_i$ equals the bit string 00000101, hence the encoded packet $\mathbf{x}_i$ has been generated by combination of the plaintext packets $\mathbf{p}_1$ and $\mathbf{p}_3$.

The introduced network coding paradigm has recently been applied to various applications, such as peer-to-peer data streaming or WSN code image update to name only a few. Obviously, there is a growing need to enrich such novel concepts with security means. Solutions regarding the integrity and authenticity of incoming encoded packets have already been proposed in [12,4,5]. Nevertheless, there is currently almost no work which explicitly deals with the confidentiality of encoded data. Exceptions are [3,1], the latter is the early version of this work. Two reasons may explain this: one can argue that the encoding in itself is already a weak mean of concealing the data and therefore no more additional protection is required. Examples following this direction are [4,6] assuming that the attacker can eavesdrop only on a subset of all transmission paths between source and destination (s). One can also argue that there is no research challenge in weaving confidentiality into the network coding paradigm by applying a cipher E: the only decision to be taken is whether to encrypt on the plaintext data or to encrypt on the encoded data. It is further possible to transmit either $(E(\mathbf{x}_i,\mathbf{c}_i))$ or $(E(\mathbf{x}_i),\mathbf{c}_i)$, or to apply the encryption on the plaintexts already $E(\mathbf{p}_i)$ before generating the $\mathbf{x}_i$. We state that all approaches, including ours, have their drawbacks either with respect to security, CPU investment or with respect to the flexibility for generating new encoded packets on its way to the final multicast destinations.

Our contribution: The contribution of this work is to conceal respectively obfuscate the data stream of network encoded packets, while allowing intermediate nodes to generate new encoded

☆ The work at hand is an extended version of the results presented at the IEEE Percom'10 Workshop SESOC [1].

* Corresponding author. Tel.: +49 40428758183.
E-mail addresses: Hessler@neclab.eu (A. Hessler), Kakumaru@neclab.eu (T. Kakumaru), Heiner.Perrey@haw-hamburg.de (H. Perrey), Westhoff@informatik.haw-hamburg.de (D. Westhoff).

0140-3664/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.comcom.2010.11.004

packets based on already received ones. This is done in a lightweight manner by purely hiding the information about the composition of the encoded packets. Examples where we see value for our solution are environments with restricted devices and/or in cases where energy saving of security enabled devices due to eco-IT aspects is of relevance. This may be a desirable feature particularly for wireless mesh networks. We evaluate our results with respect to two types of data streams: MPEG-2 video streams and Java bytecode.

2. Related work

Cai and Yeung introduced the problem of using network coding to achieve perfect information security against a wiretapper who can only eavesdrop on a subset of the transmission paths between source and destination(s) [4]. Feldman et. al. generalized and simplified the method by showing that the problem is equivalent to finding a linear code with certain generalized distance properties [5]. Tan and Medard have investigated a network coding scheme that has both a low network cost and a low success probability of the wiretapper [6]. Other improved solutions against a wiretapper have been proposed in [8,9]. However, in a wireless network scenario like we are assuming, an eavesdropper may collect packets which have been sent by the source. There have been other previous pieces of work relevant to the security of network coding which has been focused on pollution, in which the attacker nodes inject bogus packets in the network [11,13]. These contributions are mainly addressing the authenticity of the data sent by a legitimate sender, but do not provide any confidentiality. Jaggi et.al. introduced a solution that works in the presence of Byzantine nodes [14]. Ho et. al. has also referred to Byzantine modification detection [15]. There have been several research regarding a trade-off between network cost and network vulnerability for multipath traffic in wireless *ad hoc* networks [16,17]. When the link is to be resilient against wiretapping, multiple disjoint paths may be used. In general, it causes an increase of the network cost.

A similar approach to ours has been presented by Fan et al. [3]. They also propose to encrypt the coefficient vector. However, contrarily to our approach they use a public key based privacy homomorphism. Moreover, their focus is on convergecast traffic, where sensors report data to a sink node. The solution has the advantage to also provide source anonymity, since the source ID is masked as it is implicitly contained in the coefficient vector. However, their solution is not very practical, as the privacy homomorphism which they rely on produces a very large cipher. Hence, the encrypted coefficient vector that is sent over the network is growing to a few mega byte.

3. Network and adversarial model

3.1. Network model

The goal of the network is to transmit large bulks of data (i.e. larger than hundredfold the MTU size) which are fully available at the sender side in a multi-hop manner to a multicast group. The network medium is considered to be noisy and highly error-prone. This is why we believe our solution is most promising in wireless environments. The network model consists of numerous (wireless) nodes distributed forming a connected graph. We assume that the set of nodes is partitioned into three distinct roles: source, forwarder, and receiver. We assume that during the transmission of the data, the receivers remain connected to the source, possibly over several hops. There can be several sources, and forwarders can come and go. However, in this work we will keep a model with only one source, and the network remains static. We assume a flow of data that comes from the source to the set of receivers, and possibly relayed by the set of forwarders. A forwarder distinguishes itself from a receiver by the fact that it is not interested in the data, but cooperatively forwards it, such that eventually all the receivers completely collect the data transmitted by the source. Furthermore, a routing overlay has to be available such that the diffusion of the data can be achieved efficiently. Such an overlay is build upon an interest for the data transmitted by the source. To enhance the resiliency against eavesdroppers, a multipath message propagation is advisable.

3.2. Adversarial model and security requirements

We assume that the attacker is omnipresent and thus is in complete control of the wireless channels over the whole network. She can eavesdrop packets over the wireless broadcast medium or control the communication channel to delete, modify, and send data. However, she cannot capture nodes from the receiver or the source sets. As the forwarders do not have to store any sensitive cryptographic material, their capture is irrelevant in our model. In this sense, our adversary model is in line with the classical Dolev-Yao threat model [2]. Finally, an attacker should not be able to interfere with the decoding process infinitely. We assume to have a mechanism to recover from injection attacks: Once she stops attacking actively the network, the receivers shall be able to decode correctly the source data.

The security goal we are aiming at in this work at hand is to *obfuscate the data* while minimizing computational and transmission overhead. Of course, other means like ensuring authenticity and integrity of the encoded data are also important. An exemplary approach for the latter protection aim is described in [12].

4. CNC – concealed network coding

Our approach which we termed *Concealed Network Coding* (CNC) is simple but meaningful: we propose to encrypt the coefficient vector instead of the encoded packet such that $(\mathbf{x}, E(\mathbf{c}))$. To avoid giving up any flexibility regarding the composition of new and meaningful encoded packets on its way to the final destination, E should belong to a specific class of encryption transformations. It should belong to the encryption transformation class of privacy homomorphisms, such that it holds $E(a \oplus b) = E(a) \otimes E(b)$ for any plaintext pair a and b, a properly chosen additive operation $\otimes$ on the ciphertext, and another additive operation $\oplus$ on the plaintext. The benefits for such a construct $(\mathbf{x}, E(\mathbf{c}))$ are manifold:

- Although the encoded payload $\mathbf{x}$ is not directly encrypted, the fact that the coefficient vector is encrypted provides obfuscation for $\mathbf{x}$ too; Recall that $\mathbf{x}$ is a random combination of d plaintext packets neither providing any information about the concrete d nor revealing the chosen plaintext packets denoted as $\mathbf{p}_1, \ldots, \mathbf{p}_d$ without loss of generality.
- Although the encoded payload $\mathbf{x}$ is implicitly concealed, still meaningful and syntactically correct network coding on two (or more) encoded and encrypted packets $(\mathbf{x}_i, E(\mathbf{c}_i))$ and $(\mathbf{x}_j, E(\mathbf{c}_j))$, can be performed: $(\mathbf{x}_f, E(\mathbf{c}_f)) = (\mathbf{x}_i \oplus \mathbf{x}_j, E(\mathbf{c}_i) \otimes E(\mathbf{c}_j))$; the operation $\oplus$ is the concrete encoding operation, e.g. in the easiest case the bitwise XOR operation, which is in fact an addition in the vector space $\mathbf{F}_2^n$; however, it is essential to point out that other operations may also be possible.
- In a setting with $E(\mathbf{x}, \mathbf{c})$, such an in-network coding approach on encrypted data is only possible if $E()$ is homomorphic respectively to the operation $\oplus$. This property is not achieved by many ciphers.
- Since $\mathbf{c}$ is encrypted, the attacker cannot modify encoded packets at will, she lacks the knowledge of which packets she is combining together.

Please cite this article in press as: A. Hessler et al., Data obfuscation with network coding, Comput. Commun. (2010), doi:10.1016/j.comcom.2010.11.004

- Finally, please note that encrypting/decrypting a small **c** is less CPU consuming than encrypting a much larger encoded packet **x**. It requires $\frac{|\mathbf{c}|}{|\mathbf{x}|}$th of the ciphering costs if the homomorphic encryption and decryption functions $E()$ and $D()$ cause similar CPU consumption than e.g. an RC4 based ciphering.

Using this privacy homomorphism based encryption approach for obfuscating network-coding based multicast traffic, an intermediary node can still apply network coding on the packets it forwards, but has no indication of what it is transmitting.

Let us define the privacy homomorphism transformation as $E(\mathbf{p},k)$ and its associated decryption operation as $D(\mathbf{e},k)$. Here **p** and **e** represent the plaintext and the encrypted text respectively, and k denotes the key. For an instance of k and **p**, it holds that: $\mathbf{p} = D(E(\mathbf{p},k),k)$. Furthermore, it holds: $\mathbf{p}_1 \oplus \mathbf{p}_2 = D(E(\mathbf{p}_1,k) \otimes E(\mathbf{p}_2,k))$.

Now at the intermediary node, a node can combine two or more encoded packets together thanks to the privacy homomorphic transformation. For example, if the intermediary node had received: $(\mathbf{x}_1, E(\mathbf{c}_1,k_1))$ and $(\mathbf{x}_2,E(\mathbf{c}_2,k_2))$, it can combine them into another encoded packet $(\mathbf{x}_1 \oplus \mathbf{x}_2,E(\mathbf{c}_1, k_1) \otimes E(\mathbf{c}_2,k_2))$; please note that the notation indicates that we are applying a symmetric homomorphic encryption scheme using pairwise keys. However, the proposed concept also works with other homomorphic encryption schemes, e.g. symmetric and groupwise keys but also asymmetric homomorphic schemes. Various homomorphic encryption transformations have been discussed in the literature. For example, a symmetric one with pairwise keys has been proposed by Castelluccia, Mykletun and Tsudik [18]. A candidate for an asymmetric additively homomorphic scheme is the EC-ElGamal [19]. An overview on available schemes can be found in [10].

4.1. A proposed derivate

For a concrete setting of *CNC* we propose to apply the symmetric additively privacy homomorphism (PH) from Castelluccia et al. [18] with $E_k(c) = c + k \mod n$ and $D_k(E_k(c)) = E_k(c) - k \mod n$ where $c \in [0, n-1]$. However, instead of applying it on the ring $\mathbb{Z}/n\mathbb{Z}$ with the + operator like in [18], we apply the transformation on the vector space $\mathbf{F}_2^n$ with $\oplus$. To build this PH, we need to produce a key stream using a stream cipher. To ensure that no pair of encrypted vectors is using twice the same keystream, the sender applies for each encryption a unique initialization vector *IV*. For this CNC derivate the source node encrypts the coefficient vector $\mathbf{c}_i$ of each encoded packet with a different initialization vector IV_i for all transmitted packets such that the packet content is $\{\mathbf{x}_i, E_{k,IV_i}(\mathbf{c}_i), IV_i\}$. An intermediate node that generates a new encoded packet $\mathbf{x}_f$ from two (or more) encoded packets $\mathbf{x}_i$ and $\mathbf{x}_j$ is computing $\mathbf{x}_f = \mathbf{x}_i \oplus \mathbf{x}_j$ and $E_{k,IV_i,IV_j}(\mathbf{c}_f) = E_{k,IV_i}(\mathbf{c}_i) \otimes E_{k,IV_j}(\mathbf{c}_j)$. Subsequently, the intermediate node transmits $(\mathbf{x}_f, E_{k,IV_i,IV_j}(\mathbf{c}_f), IV_i, IV_j)$. The initialization vectors can tell the attacker how many times encoded packets were combined. This is not an issue, as we do not assume that the degree *distribution* is secret. At the final receiver side (one node out of the multicast group), an incoming message $\mathbf{x}, E_{k,IV_1,\ldots,IV_l}(\mathbf{c}), IV_1, \ldots, IV_l$ is decrypted before the decoding starts: $\mathbf{c} = D_k(E_{k,IV_1,\ldots,IV_l}(C))$. The decryption process is simply to generate all the keystreams that have been used by the sender, using the received *IV*s, and XOR them to the ciphertext. The keystreams are cancelled out such that the coding vector **c** is revealed.

Please note that originally the PH from Castelluccia et al. was proposed in a setting for convergecast traffic with the number of keys corresponding to the size of the convergecast group. In such a setting the list of key Ids to be transmitted could easily explode. However, in our setting for multicast traffic the size of the *IV* list (l) corresponds to the number of combined encoded packets that originate from the source, such that it will always be reasonably small.

5. General remarks on CNC

We start the discussion on CNC by providing some general remarks with respect to its overhead, the decoding process itself, the security parameters, the ratio of $|\mathbf{x}|$ and $|\mathbf{c}|$, and the impact of blind aggregation respectively decoding on the fountain code. Table 1 gives an overview for the most relevant variable used in the text.

5.1. Network overhead

Let us assume that a group of receivers wants to extract the data stream $\mathbf{p}_1, \ldots, \mathbf{p}_n$. To be able to successfully perform the decoding process, each receiver must have received $n + \Delta$ encoded packets. Δ is the fountain code packet overhead, which is listed in Table 3. Thus, the source has to transmit in average *more* than $n + \Delta$ encoded packets $\mathbf{x}_i$, to account for the network losses that might occur.

5.2. Decoding

Decoding (without the proposed encryption approach) works by solving a linear equation system $X = CP$. C needs to be a $m \times n$ matrix of n linear independent coefficient vectors **c**, i.e. $rank(C) = n$. X is the matrix composed of the m received encoded packets **x**, and P is the matrix of cleartext packets, each row relating to one cleartext packet. In the case of LT codes [20], the decoder is using a belief propagation decoder instead of the Gaussian elimination technique. Therefore, as it requires decoded packets to further decode encoded packets, the degree distribution should contain low degree packets, of which a few are of degree one. Hence, the degree distribution is often some derivate of the ideal soliton distribution. The LT decoder has linear complexity in regards to n, but has more packets overhead (Δ). With our concealment approach an attacker has no information about the matrix C, thereby trying to infer P out of X with no additional information.

5.3. Security parameters

Since the proposed CNC approach is lightweight, it has some potential weaknesses. The attacker has some means to break the system independently of the concrete security strength of the chosen underlying homomorphic encryption transformation. Parameters of relevance are.

(i) the concrete value of the degree d (the number of cleartext packets that are used to generate a specific encoded packet) or more generally the chosen degree distribution Ω to encode the packets;
(ii) the entropy respectively the redundancy of the cleartext data;

Table 1
Overview table for used variables.

Variable	Explanation
p	Plaintext packet
x	Encoded packet
c	Coefficient vector
d	Degree of encoded packet
Ω_l	Degree distribution at location l
E, D	(Homomorphic) encryption resp. decryption function
X	Matrix composed of m encoded packets **p**
C	$m \times n$ matrix of n linear independent coefficient vectors **c**
P	Matrix of cleartext packets
$\tilde{\mathbf{x}}, \widetilde{X}$	Bogus data chunk, bogus set of data chunks
IV	Initialization vector

(iii) the number of cleartext packets n into which the cleartext data stream is subdivided. A more thorough analysis follows in Sections 6 and 8.

Please note that with such an approach of merging two or more encoded packets $\mathbf{x}_i$ and $\mathbf{x}_j$, the resulting encoded packet always contains either one or zero plaintext contribution per plaintext but definitively not more. This comes due to the fact that the XOR operation cancels out an even number of the same bit-stream (plaintext packet). Our encryption approach on the coefficient vector supports such an encoding characteristic since the homomorphic operation on encrypted coefficient vectors represents this.

5.4. Impact of $|\boldsymbol{x}|$

Generally, the size ratio is $|\mathbf{x}| \gg |\mathbf{c}|$; The size $|\mathbf{c}|$ reflects the number of plaintext packets in which the whole data stream is subdivided whereas the limiting factor of $|\mathbf{x}|$ basically is limited by the MTU of the network or other parameters limiting the packet size (e.g. Ethernet 1500 byte, e.g. 1480 byte IPv4 data). One can also consider settings, e.g. in WSNs with IEEE 802.15.4 and a huge data stream to be transmitted; here, since $|\mathbf{x}| \approx 40$ byte the above ratio is much smaller.

5.5. Impact of blind aggregation on the fountain code

We want to point out that in a multi-hop scenario with in-network encoding on received packets $(\mathbf{x}_i, E(\mathbf{c}_i))$ the degree distribution chosen at the source node Ω_{source} will change over the various hops. This is true since for an intermediate node the encoding is fully random. After k combinations of already encoded packets, the degree distribution can be denoted as

$$\Omega_k(d=w) = \sum_{i=0}^{n}\sum_{j=0}^{n} \Omega_{w-1}(d=i)\cdot\Omega_{source}(d=j)\cdot P_{i,j}^{n}(d=w), \qquad (1)$$

where $P_{i,j}^{n}(d=w)$ is the probability of obtaining a packet of degree w when combining two random packets of length n and degrees i and j, derived from a hypergeometric distribution:

$$P_{i,j}^{n}(d=w) = \begin{cases} 0 & \text{if } w < |i-j|, \\ 0 & \text{if } w > n-|i+j-n|, \\ 0 & \text{if } |i-j|-w \text{ odd}, \\ \dfrac{\binom{\max(i,j)}{\frac{i+j-w}{2}}\cdot\binom{n-\max(i,j)}{\min(i,j)-\frac{i+j-w}{2}}}{\binom{n}{\min(i,j)}} & \text{otherwise.} \end{cases} \qquad (2)$$

Fig. 1 shows the degree distribution "degeneration" with the number of original encoded packets combinations, with an initial Ω_{source} = 'ideal soliton distribution' at the source with $Pr(d=1)=\frac{1}{n}$ and $Pr(d=k)=\frac{1}{k(k-1)}$. After a few hops and random combinations performed by forwarder nodes, we end up with a degree distribution that approaches the binomial distribution $Pr(d=k)=\binom{n}{k}p^k(1-p)^{n-k}$ with $p=\frac{1}{2}$. We conclude that blind aggregation from the forwarders in CNC is inappropriate with a LT decoder.

6. Security analysis

Our security analysis distinguishes two types of attackers: Attacker A purely eavesdrops the system and has no information other than the $\mathbf{x}_1,\ldots,\mathbf{x}_m$ packets that she eavesdropped. Here m is at most the number of packets sent by the source. We consider that the attacker has always captured enough packets for the decoding process to succeed. Attacker B tries to break the system with 'some' additional information.

6.1. Attacker A

Let us consider a data stream subdivided into n plaintext packets and let d_i be the degree chosen for any concrete encoded packet $\mathbf{x}_i$. To break our system the attacker is required to solve the linear equation system $X = CP$ without knowing the matrix C. Moreover, for each row i of the $n \times m$ matrix, the probability that a guessed vector $\tilde{\mathbf{c}}_i$ corresponds to $\mathbf{c}_i$ for the concrete $\mathbf{x}_i$ can be expressed as the binomial coefficient:

$$Pr(\tilde{\mathbf{c}}_i \stackrel{?}{=} \mathbf{c}_i) = \frac{1}{\binom{n}{d_i}} = \frac{d_i!(n-d_i)!}{n!}. \qquad (3)$$

For a brute force attack, the probability for guessing the whole matrix at each try and thus obtaining all plaintext packets is

$$Pr(\tilde{C} \stackrel{?}{=} C) = \prod_{i=1}^{n+\Delta} \frac{d_i!(n-d_i)!}{n!}. \qquad (4)$$

Please note that even if the attacker can guess with probability $Pr(\tilde{\mathbf{c}}_i = \mathbf{c}_i)$ row i of C, she is not enabled to proof on a row-wise basis that her guess was correct. Verifying (by decoding) would only work when she correctly guesses the full matrix C. This is an important aspect since obviously, breaking the system by pure guesswork is not possible and we can infer that an attacker who has no additional knowledge than the stream of encoded packets $\mathbf{x}_1,\ldots,\mathbf{x}_m$ itself, will not be able to break our scheme other than brute force. For a C of dimension $(n+\Delta)\times n$ brute force has to be done on a possibility space with size $2^{n+(n+\Delta)}$. Even with an unrealistically small data stream with e.g. $n=10$ we end up with a chance for breaking the system of about 10^{-4}.

6.2. Attacker B

However, it is more realistic to assume that the attacker has some additional knowledge, let it be on the degree distribution Ω as well as on the structural information of the plaintext packets $\mathbf{p}_j$. More details on the latter fact and its impact on the overall obfuscation level will be provided in Section 8. We consequently assume that the attacker can recognize a packet with degree $d=1$, that is a cleartext message. Next, we discuss how this influences the provided obfuscation level of CNC.

Let us assume for the moment that the attacker already knows a pool $X_{captured}$ of m messages $\mathbf{x}_1,\ldots,\mathbf{x}_m$. In the most general case $m > n$. Furthermore we assume that an attacker can *distinguish* a plaintext packet e.g. due to its structuring or other means from other encoded packets. If this is not given we are back in the argumentation line described for attacker A. Then, as long as it holds $m < d_{min} - 1$ with d_{min} being the minimum degree in all transmitted encoded packets described by Ω_{source}, the attacker can not decode respectively deobfuscate a single transmitted packet $\mathbf{x}_i$. However, for the case $m \geqslant d_{min} - 1$ there may eventually be situations in which the attacker can decode an encoded packet and thus increase the pool to $|P_{new}| = |P_{old}| + x$ elements (Note that $|P| = n$ means the total break of the system). Based on its pool of packets, the attacker can combine packets at will using the network coding operation, and might recognize packets of degree one. If this happens, she increases the set of decoded packets $P_{decoded}$. Furthermore, she can use the packets from $P_{decoded}$ to further "blindly" decode packets in $X_{captured}$. In particular degree distributions Ω_{source} like 'ideal soliton distribution' are vulnerable for such kind of attacks, since many packets have low degrees. The 'robust soliton distribution' is even more vulnerable, as it contains more unencod-

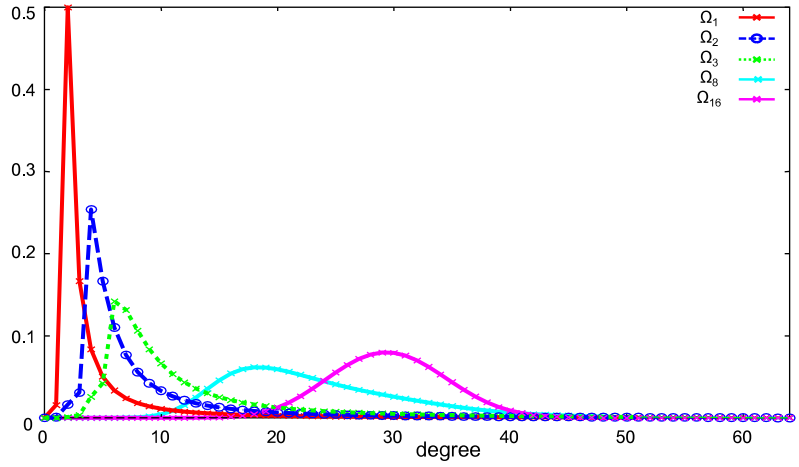

Fig. 1. Degree distribution degeneration of encoded packets due to in-network combination ($n = 64$).

ed packets ($d_i = 1$). Fig. 2 shows the amount of packets that can be determined by an attacker considering variable sizes of n and K. One can notice that by combining as few as $\binom{m}{K} \ll 2^n$ packets together, the attacker can find out about 70% of the packets with $n = 64$, $m = 96$ and $K = 6$. K denotes the maximum number of packets being combined together. Furthermore, the attacker can use this pool of decoded packets to continue its attack. The attack shown in Fig. 2 only uses eavesdropped packets, and does not take advantage of newly decoded packets. So it is a lower estimate of the capacity of the attacker.

However, if the source degree distribution is binomial, then this attack is impractical. Combining two or more packets from this distribution yields a packet with the same binomial degree distribution. Therefore, obtaining randomly a source packet is extremely unlikely and makes this attack unprofitable for the attacker. Nevertheless, such a source distribution is not the panacea, as it has a negative impact on the decoder performance. The main lesson learned is that Ω_{source} and the structural information of the source data have an impact on the obfuscation level provided by CNC.

Remark: From our general remarks in Section 5 on the *impact of blind aggregation* on the fountain codes we know that in a multi-hop data propagation scenario the degree distribution anyhow results in a binomial degree distribution. To ensure more resistance against attacker B (which may eavesdrop at the optimal location directly at the source node) and due to the fact that anyhow the receiving nodes in presence of CNC have to decode based on a binomial degree distribution, we conclude to also choose Ω_{source} = 'binomial degree distribution' at the source node.

6.3. Additional countermeasures

In addition to the proper choice of Ω_{source}, we observe that we have four countermeasures for increasing the obfuscation level, each with its different pros and cons:

(i) *Disallow low degree packets*: choosing a Ω with a threshold T_{drop} s.t. $Pr(d_i) = 0 \, \forall i \leqslant T_{drop}$ which is significantly higher than one, e.g. $T_{drop} = 3$ or $T_{drop} = 4$; Such a choice impacts the CPU consumption for the decoding process at the receiver side.

(ii) *Encrypt low degree packets*: encrypting all encoded packets with degree $d_i \leqslant T_{enc}$ as follows: $E(\mathbf{x}_i, \mathbf{c}_i)$. Thus, only a fraction of all encoded packets are encrypted in this traditional way. This reduces considerably the initial pool $P_{decoded}$ of the attacker and therefore increasing the overall security level for all transmitted encoded packets (also with $d > T_{enc}$) significantly. At the same time there is only a marginal increase of overhead due to the amount of data to be transmitted for sending and receiving (one could use a normal streamcipher for the encryption of all packets with degree $d \leqslant T_{enc}$) by still being able to encrypt for a large fraction of encoded packets,

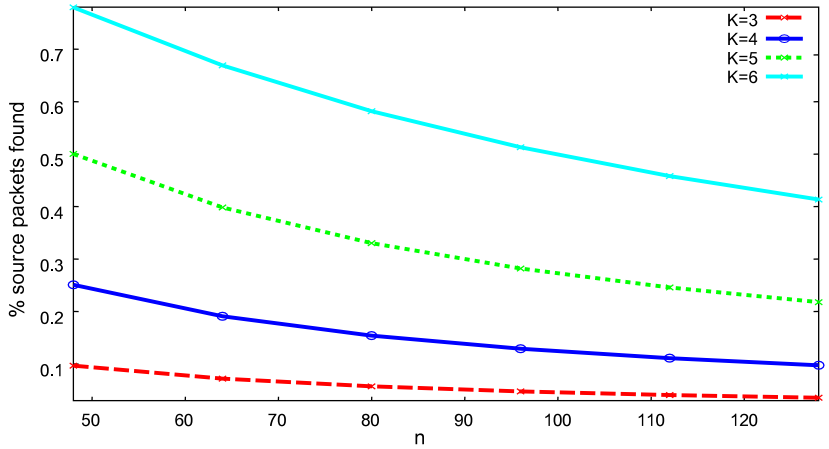

Fig. 2. Brute force attack with attacker B: obtaining source packets from combining eavesdropped packets together. ($m = 1.5 \cdot n$).

Table 2
Location of overhead for CNC extensions and their impact on the obfuscation robustness.

	Source	Forwarder	Receiver	Obfuscation
(i) $d_{min} \geqslant T_{drop}$	–	–	(cpu)	+
(ii) $E(\mathbf{x}_i)$ with $d_i \leqslant T_{enc}$	(cpu/msg)	(msg)	(cpu/msg)	++
(iii) ϵ packets $\tilde{p}$	(msg)	(msg)	(msg)	+
(iv) Input obfuscation	(cpu)	–	(cpu)	+

namely $Pr(d \leqslant T_{enc}) = 1 - Pr(d > T_{enc})$, only the coefficient vectors and being flexible for in-network processing. Recall that in most settings $|\mathbf{x}| \gg |\mathbf{c}|$.

(iii) *Increase attacker decoding cost*: make use of well marked, well structured *bogus* plaintext packets $\tilde{\mathbf{p}}$. This is a pollution attack against the attacker. The source injects a portion of encoded packets $(\tilde{\mathbf{x}}_i, \mathbf{c}_i = E(\text{'}do\_not\_use\text{'}))$ that cause noise for the attacker but can easily be filtered out at the receiver side by decrypting the 'coefficient vector'. Encoded packets which are classified with a *do_not_use* tag will not go into the equation system $X = CP$ such that the receiver can be assured (recall that we are only discussing *passive* attackers) that he can successfully decode the linear equation system if it holds $rank(G) = n$. On the other hand the attacker cannot differentiate a cheated plaintext packet $\tilde{\mathbf{p}}$ from a plaintext packet belonging to $\mathbf{p}_1, \ldots, \mathbf{p}_n$. Consequently, after a while her pool will be filled up like $P_{decoded} = \{\mathbf{p}_1, \ldots, \mathbf{p}_{r-1}\} \cup \{\tilde{\mathbf{p}}_r, \ldots, \tilde{\mathbf{p}}_m\}$ whereas the genuineness of packets $\mathbf{p}$ and $\tilde{\mathbf{p}}$ are undistinguishable for the attacker. Recall that the coefficient vector of each $\mathbf{x}_i$ is encrypted such that the attacker does neither know the concrete plaintext packets which have contributed nor does she know the concrete degree d of the actual packet. Under such circumstances if only one single packet $\tilde{\mathbf{p}}$ will be part of an decoding attempt for an $\tilde{\mathbf{x}}_i$ decoding will always fail. This approach requires the transmission of $n + \Delta + \epsilon$ packets, therefore increasing the sending and receiving effort at any involved node. Furthermore, as the forwarder nodes cannot identify bogus packets, further encoding by the forwarders is not possible here. However, the pure decoding process at the final receiver nodes is not increased (excluding the decryption and subsequent skipping of packets $(\tilde{\mathbf{p}}_i, E(\mathbf{c}_i))$).

(iv) *Input obfuscation*: we have seen previously how easy it is for an attacker to break CNC if low degree of encoding and recognizable source data are employed. To thwart this, we could pre-code the input symbols by applying an outer code to spread out over the input symbols. It spreads out the structured input symbols over all the symbols by bit-wise spread code and hence cannot distinguish a plaintext packet. Compressing the source data before the CNC transmission is also a good counter-measure to prevent statistical attacks as we will see.

Table 2 summarizes for which node's role the overhead of the proposed solutions have an impact on.

Table 3 summarizes the different network coding variants with the overhead and obfuscation parameters [21]. Not surprisingly, there is a trade-off between decoding efficiency and the provided obfuscation level. For more explanation, we refer the reader to the Appendix A.

7. Benefits of CNC

Our analysis regarding the CNC approach and its variants is following three axes: *computation* effort, *security* level, *transmission* overhead.

- *computation effort*: With some simplification we can state that the larger the ratio $|\mathbf{x}|/|\mathbf{c}|$, the more advantageously is CNC compared to a traditional encryption approach. However, we want to point out that this statement only holds under the assumption that the chosen homomorphic encryption function has similar CPU consumption as e.g. the stream cipher RC4. This statement holds for the chosen symmetric homomorphic encryption transformation.
- *security level*: Obviously the security level of CNC cannot compete with a traditional symmetric stream-cipher e.g. RC4 or a block-cipher e.g. AES based encryption of $E(\mathbf{x}, \mathbf{c})$. However, in particular when enriching our approach with one of the proposed extensions, the achieved obfuscation level is by far more than 'better-than-nothing' security. We will provide some more insight with respect to the obfuscation level provided by CNC in Section 8.
- *transmission overhead*: The overhead for the transmission costs when applying pure CNC is equivalent to the data overhead when encrypting the full packet in a traditional way. Both approaches, pure CNC and the traditional one, are causing approximately 10% overhead depending on the chosen security parameter, e.g. with a data size of 1 KB and a MTU of 40 B. However, for the extension (iii), which is injecting packets, we have to face additional transmission overhead. For the extension (ii), which consists in encrypting low degree packets, there is less transmission overhead because of low rate of degree one. Otherwise, for the other extensions, there is no transmission overhead.

In Fig. 3, we display the results for the case the data size to be transmitted is 1 Kbytes, the MTU is 40 bytes and the IV is three bytes. A setting that represents WSNs. It shows that the size of the transmissions made in CNC including security extentions (i) and (iv) is about 10% bigger compared to fountain codes with no security, however equivalent to the size where encrypting the whole packet in a traditional way except for cases of different number of packets to be transmitted because CNC could be adapted by compressing multiple IVs. Furthermore, it also shows that applying extension (iii) is 1.5 times larger compared to CNC. On the other hand, extension (ii) is slightly larger than CNC because of the low rate of degree one packets. Although we study only the transmission cost, there is also a corresponding receiving cost associated to every transmission and each receiver.

The discussion of these three axes shows that CNC, together with its extension (ii) that encrypts low degree packets and extension (iv) that obfuscates the source data, is a good choice in all settings where the encoded part of a packet is significantly larger than

Table 3
Various fountain codes techniques.

	Encoding complexity	Decoding complexity	#packets	Obfuscation with CNC
Send As Is	$O(1)$	$O(1)$	n	None
Raptor Codes	$\Theta(1)$	$O(n)$	$(1+\epsilon)(1+\epsilon')n$	+
Unif. rand.	$\Theta(n)$	$\Theta(n^3)$	$n + \Theta(\log n)$	++
LT Codes	$O(\log n)$	$\Theta(n \log n)$	$n + \Theta(n)$	+

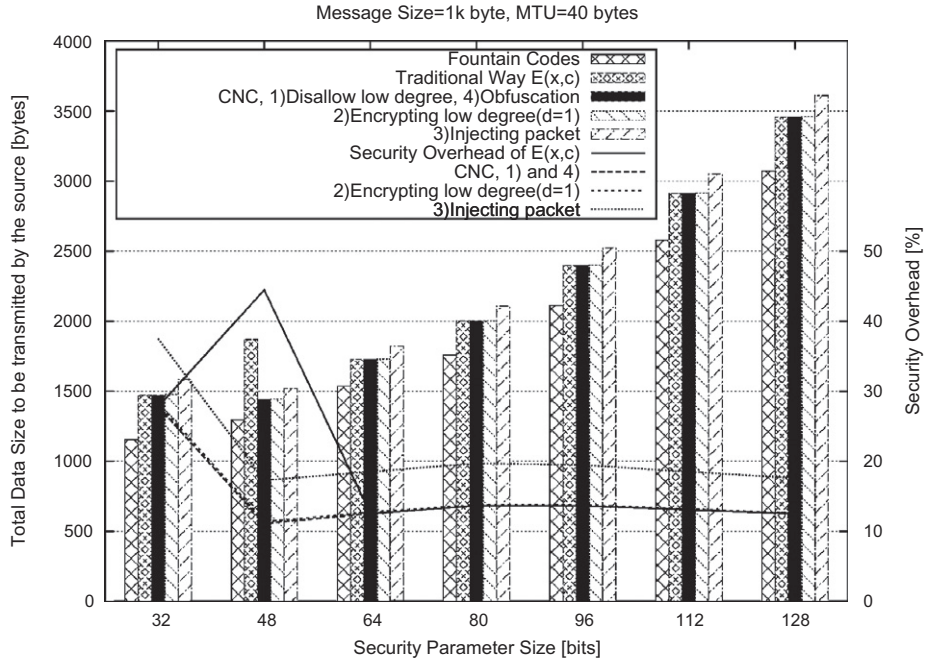

Fig. 3. Communication costs (in bytes) and security overhead comparison when 1k byte data is transmitted by the source and MTU on a layer of a communications protocol is 40 bytes.

the coding vector itself. Here, CNC has a clear benefit, in particular since the reduced CPU-costs at the source and receiver side for encryption and decryption are not overcompensated due to overhead for sending and receiving of additional data. Clearly, since at this moment we cannot provide a proper analysis of the achieved obfuscation level but only educated guesswork, we recommend our solution only in settings where weak respectively moderate security is required. In all other settings the traditional encryption approach is preferable.

8. Impact of recognizable data

This section revisits the implicit pre-assumptions we made in the security analysis of Section 6 with respect to the attacker A and the attacker B and which we believe is worth being analyzed more in depth: *an encoded packet* $\mathbf{x}$ *does not provide enough structure to infer the contributing plaintext packets* $\mathbf{p}$. For the attacker A one pre-assumption even means that she *can not distinguish a packet* $\mathbf{p}$ $(d = 1)$ *from a packet* $\mathbf{x}$ $(d > 1)$ *by analyzing their structuring.*

8.1. Preliminaries

We start our discussion by looking at MPEG-2 video streams: Regarding the structural information of an intra-coded (I)-frame, an predictive-coded (P)-frame or a bidirectionally-predictive-coded (B)-frame, we observe that each frame contains a structured part with MPEG-2 signaling data and the payload part containing relatively unstructured information. An unstructured sequence is a series of bits whose values follow a probability distribution close to the uniform one. For the MPEG-2 file, this property is provided by the compression of the image blocks and motion vectors. We will show that with a properly chosen degree d (from a properly chosen degree distribution Ω for the fountain encoding) and some shifting operation if required, one can almost nullify these structural information for an encoded packet $\mathbf{x}$ such that without any additional knowledge an encoded packet $\mathbf{x}$ with $d > 2$ does not reveal any information with respect to its content, respectively the involved packets $\mathbf{p}$. Obviously, this additional information comes with the coefficient vector $\mathbf{c}$.

To clarify the above statement consider two cleartext packets $\mathbf{p}_1$ and $\mathbf{p}_2$. Without loss of generality and independently if the concrete packets consist of I-frame, P-frame or B-frame chunks we can denote a cleartext packet as $\mathbf{p} = s \ldots su \ldots us \ldots su \ldots u$ where the s is a bit of a structured bit-sequence and the u is a bit of an unstructured bit-sequence. We observe that an encoding operation of degree $d = 2$, namely $\mathbf{x} = \mathbf{p}_1 \oplus \mathbf{p}_2$ results in an encoded sequence with bits $s = s \oplus s$, $u = s \oplus u$, $u = u \oplus s$ and $u = u \oplus u$. As an example consider the encoding of two unrealistically small one byte-plaintext packets:

$$\begin{aligned} \mathbf{p}_1 &= ssuuuuuu, \\ \underline{\oplus \mathbf{p}_2} &= \underline{ssuussuu}, \\ \mathbf{x} &= ssuuuuuu. \end{aligned} \tag{5}$$

Obviously an u-bit is *dominant* and valuable with respect to our request in the sense that it nullifies an s-bit. This characteristic becomes even more significant with any $d > 2$. By XORing $d - 1$ s-bits with a single corresponding u-bit always results in an u-bit in the resulting encoded packet $\mathbf{x}$.

We observe that fountain codes in itself provide obfuscation under the following prerequisites:

1. the coefficient vector $\mathbf{c}$ is concealed (with CNC);
2. avoid sending encoded (and unencrypted) packets of degree $d < 4$ (or $d = 2$ when compressed);
3. avoid bits s in $\mathbf{x}$ which result from $s = s_1 \oplus \cdots \oplus s_d$;

A more substantial validation on (2) and (3) will follow in the subsequent benchmark section for MPEG-2 video streams and Java bytecode, both, with compression and without compression.

8.1.1. Remark

An encoded packet $\mathbf{x}$ with $d = 2$ is precisely an expression $\mathbf{p}_1 \oplus \mathbf{p}_2$ known as the result of an attack against WLAN's first generation security protocol *Wired Equivalent Privacy* (WEP) where the

originally weaved in keystream k could be eliminated from two ciphers $c_1 = \mathbf{p}_1 \oplus k$ and $c_2 = \mathbf{p}_2 \oplus k$ such that

$$(\mathbf{p}_1 \oplus k) \oplus (\mathbf{p}_2 \oplus k) = \mathbf{p}_1 \oplus \mathbf{p}_2 \quad (6)$$

An attacker that obtains such an expression $\mathbf{p}_1 \oplus \mathbf{p}_2$, *since knowing that it is composed of exactly two plaintext packets*, has almost broken the system. Once she gets one of the plaintext packets $\mathbf{p}_1$ or $\mathbf{p}_2$ she can compute the other one and subsequently derive the key k. For more details on WEP attacks and WPA we refer to [7]. This attack clearly indicates that the CNC approach itself is required to avoid under any circumstances that the attacker is enabled to get an encoded packet $\mathbf{x}$ for which she *either knows or can derive* that its actual degree is $d = 2$.

8.2. Unstructured encoded packets

Ideally, we would like to solely transmit encoded packets of the form $\mathbf{x}_u = uuu\ldots u$. To achieve this, one can adjust the fountain encoding process at the source node with regard to two axes: One approach is to only allow transmission of encoded packets with relatively high degree d. The larger the d the higher the probability that at least one u-bit position of one cleartext packet $\mathbf{p}_i$, $i = 1,\ldots,d$ is available which would then result in an $\mathbf{x}_u$. However, since also for large d, and with respect to the decoding process an inefficiently large degree, we cannot guarantee generating an encoded packet of type $\mathbf{x}_u$ we also propose to *actively* obfuscate the packet structure at the source node. The source is aiming to

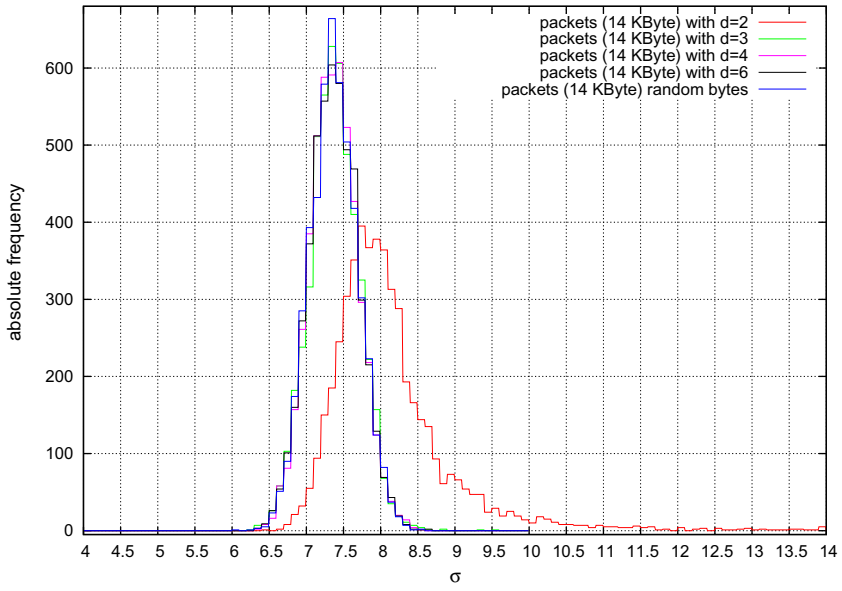

Fig. 4. Comparison of encoded packets with different degrees (no compression).

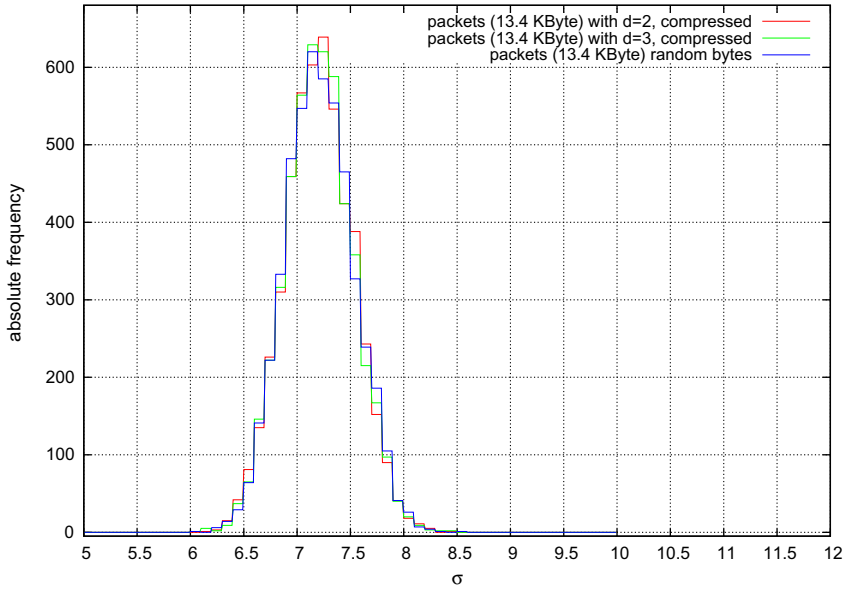

Fig. 5. Comparison of encoded packets with different degrees (compressed).

only transmit encoded packets $\mathbf{x}_u$ by applying the following operations in this order:

1. in the best case only combine such d packets which result in an $\mathbf{x}_u$ and transmit $\mathbf{x}_u$s that contain *fresh* cleartext packets $\mathbf{p}_i$ as long as this holds; always choose the proportion of ds with different size according to the given degree distribution Ω;
2. `if` cleartext-packets $\mathbf{p}_i$ remain which result in an $\mathbf{x}_s$ with respect to approach (1), `then do` $x \hookrightarrow p$ on some packets such that their encoded combination definetively performs to an $\mathbf{x}_u$; transmit them as long as this holds;
3. `if` only cleartext-packets $\mathbf{p}_i$ remain which still result in an $\mathbf{x}_s$ with respect to approach (2), `then do` $x \mapsto \mathbf{p}$ on some packets $\mathbf{p}$; transmit them until every cleartext packet was included in at least one encoded packet $\mathbf{x}$ (ideally $\mathbf{x}_u$);

The notation $x \hookrightarrow \mathbf{p}$ denotes a circular rightshift in $\mathbf{p}$ of x bits. If in our example in Eq. (5) we could shift the bits of cleartext packet $\mathbf{p}_2$ two positions to the right, we would have encoded a fully unstructured packet $\mathbf{x}_u$.

$$\begin{aligned} \mathbf{p}_1 &= ssuuuuuu, \\ \underline{\oplus(2 \hookrightarrow \mathbf{p}_2)} &= \underline{uussuuss}, \\ \mathbf{x}_u &= uuuuuuuu. \end{aligned} \tag{7}$$

However, it may turn out that the shift operation $x \hookrightarrow \mathbf{p}$ to some plaintext packets may not always help to generate an $\mathbf{x}_u$. In such cases, and only under such circumstances, we introduce the shift operation $x \mapsto \mathbf{p}$ which applies x padding bits l from the left side. For our example a $2 \hookrightarrow \mathbf{p}_2$ would result in the following encoded packet.

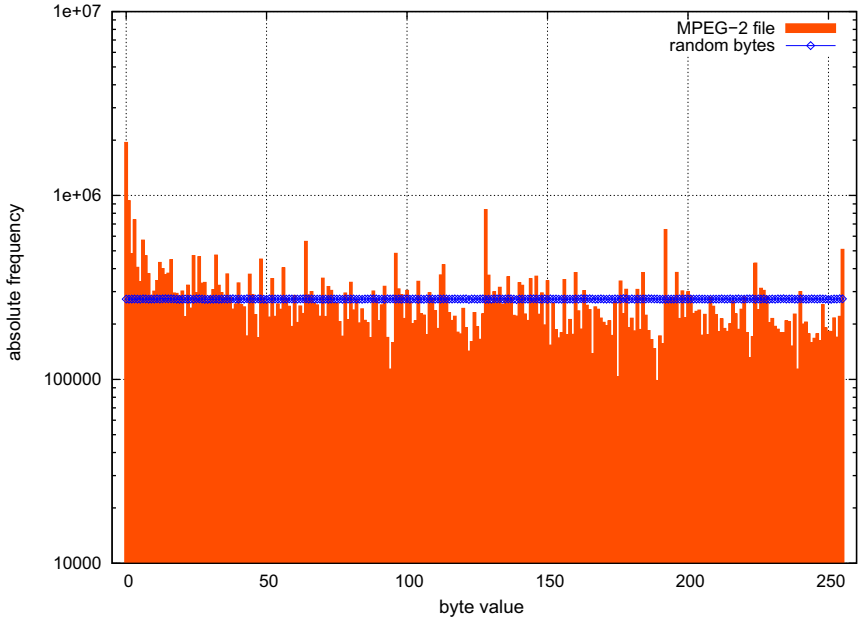

Fig. 6. Comparison of a pseudo-randomly generated file with a not encoded MPEG-2 video stream.

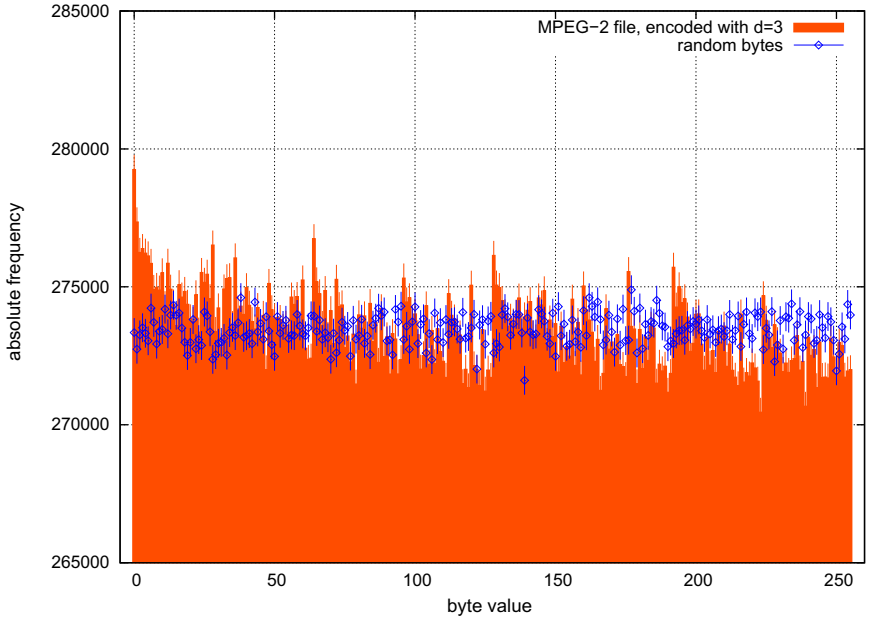

Fig. 7. Comparison of a pseudo-randomly generated file with a MPEG-2 video stream purely encoded with $d = 3$.

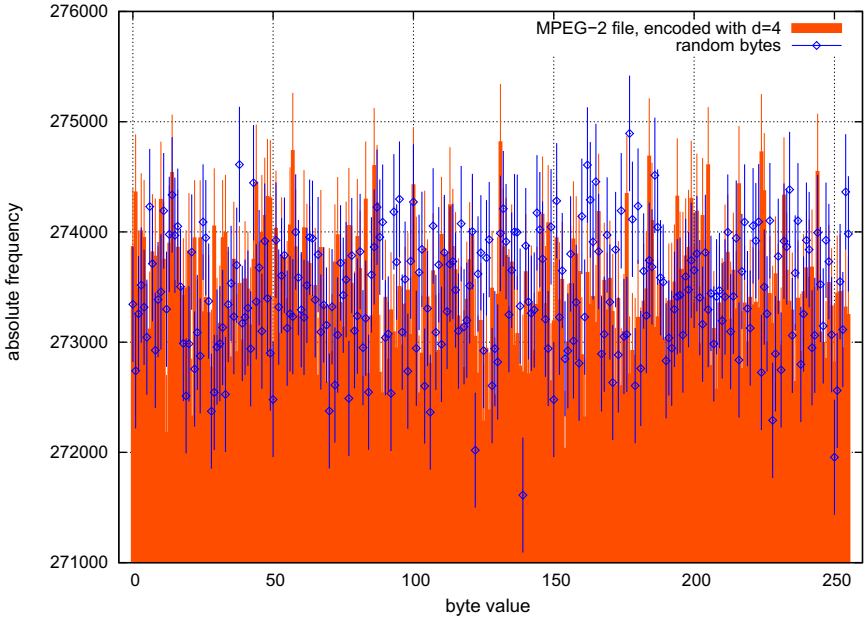

Fig. 8. Comparison of a pseudo-randomly generated file with a MPEG-2 video stream purely encoded with $d = 4$.

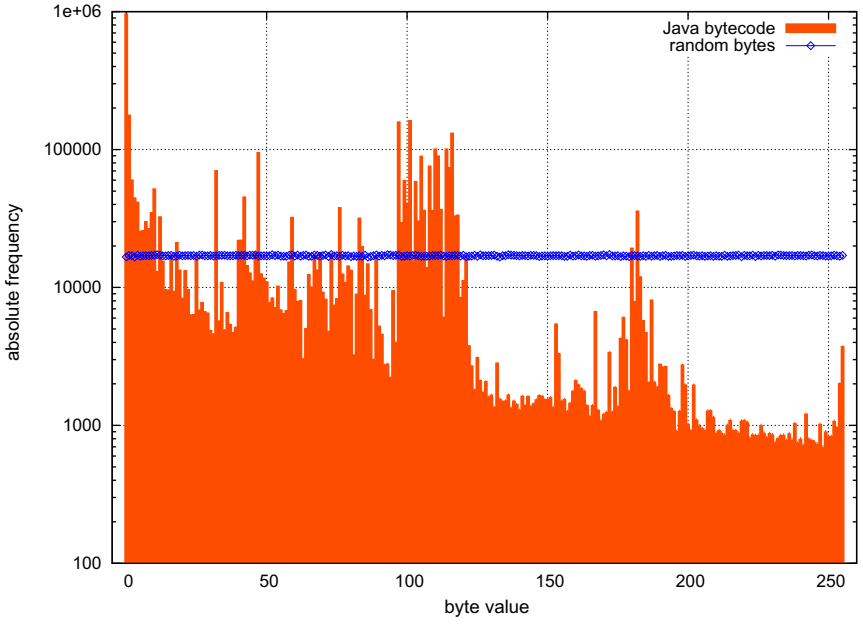

Fig. 9. Comparison of a pseudo-randomly generated file with Java bytecode.

$$\begin{aligned}\mathbf{p}_1 &= ssuuuuuu,\\ \underline{\oplus 2 \mapsto \mathbf{p}_2} &= \underline{lluussuu},\\ \mathbf{x}_u &= uuuuuuuu.\end{aligned} \tag{8}$$

Obviously the latter generation of an encoded packet is crucial and we can only classify the result as $\mathbf{x}_u$ iff the l bits are unstructured. This can only be achieved by an extension of the encrypted part, e.g. to $\mathbf{x}, E(\mathbf{c}, x \mapsto \mathbf{p}_i)$ to be able to skip the leading x bits of the plaintext $\mathbf{p}_i$ subsequently to the decoding at the receiver side.

Preferably the source node should only apply the first two approaches to generate unstructured encoded packets. Ideally, it should only apply the first approach which is appropriate for most MPEG-2 data stream properties as we will see.

Remark (cont.): We would like to remind the reader that packets of reasonable high degree alone are not enough to provide a good obfuscation, and that also a proper degree distribution is important. To illustrate this, imagine that the attacker randomly picks two transmitted packets, say both of degree $d = 3$. Let the concrete encoded packets be $\mathbf{x}_1 = \mathbf{p}_{12} \oplus \mathbf{p}_{23} \oplus \mathbf{p}_{31}$ and $\mathbf{x}_2 = \mathbf{p}_7 \oplus \mathbf{p}_{12} \oplus \mathbf{p}_{31}$. Now, the attacker's aim is to cancel out some plaintext packets and infer by *statistically analysing* the structure of the resulting $\mathbf{x}_s$ its remaining degree d_s. For a $d_s > 2$, this is not an option as our simulation results will show. However, it *may* be possible to *distinguish*

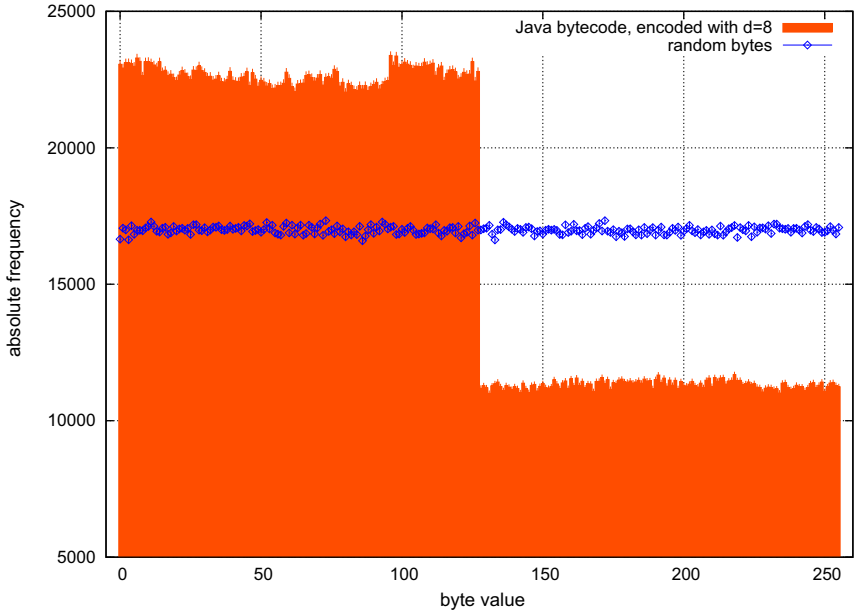

Fig. 10. Comparison of a pseudo-randomly generated file with an encoded Java bytecode ($d = 8$).

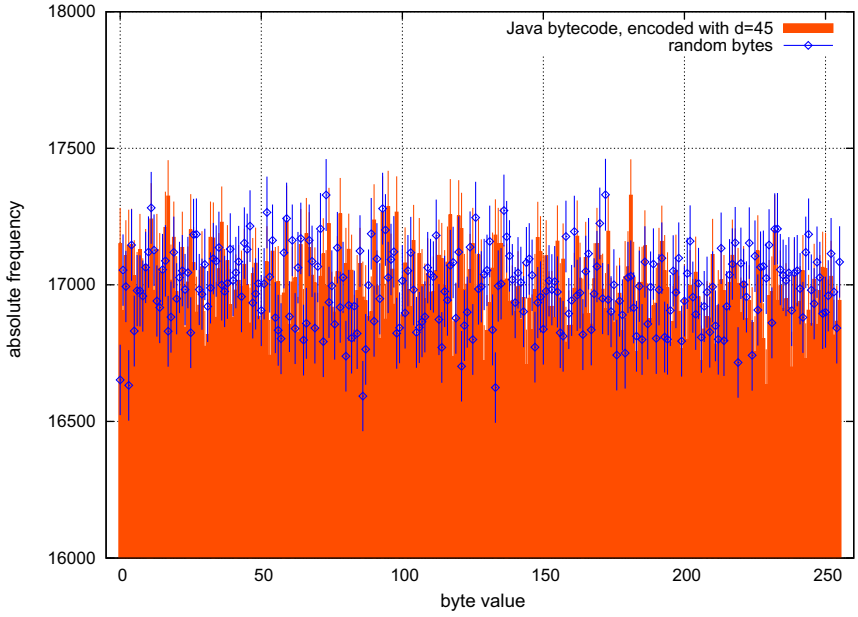

Fig. 11. Comparison of a pseudo-randomly generated file with an encoded Java bytecode ($d = 45$).

an encoded packet with $d_s = 2$ from one with a higher degree. This would then be an expression which in WEP ist considered as (almost) broken.

For our concrete example the attacker generates in analogy to the attack shown in Eq. (6) the encoded packet $\mathbf{x}_s$:

$$\begin{aligned} \mathbf{x}_1 &= \mathbf{p}_{12} \oplus \mathbf{p}_{23} \oplus \mathbf{p}_{31}, \\ \underline{\oplus \mathbf{x}_2} &= \underline{\mathbf{p}_7 \oplus \mathbf{p}_{12} \oplus \mathbf{p}_{31}}, \\ \mathbf{x}_s &= \mathbf{p}_7 \oplus \mathbf{p}_{23}. \end{aligned} \tag{9}$$

Due to the significantly different structure e.g. of a MPEG-2 encoded packets with a degree of 2 compared to packets with a degree of 3 (see Fig. 4), one could argue that the attacker *may* be able to recognize that the $\mathbf{x}_s$ is exactly of degree $d = 2$. At this point, if one packet (either $\mathbf{p}_7$ or $\mathbf{p}_{23}$) is already known to the attacker, she can also decode the other one.

However, please note that the results in Fig. 4 are based on the uncompressed data. As we will see next, according to our current analysis, for a compressed datastream it is almost impossible to recognize the actual degree of an encoded packet.

8.3. Benchmark results

In our subsequent analysis we are aiming to estimate the probability of spotting structures in the encoded files by a statistical analysis. These structures are based on single bytes. The alignment

Please cite this article in press as: A. Hessler et al., Data obfuscation with network coding, Comput. Commun. (2010), doi:10.1016/j.comcom.2010.11.004

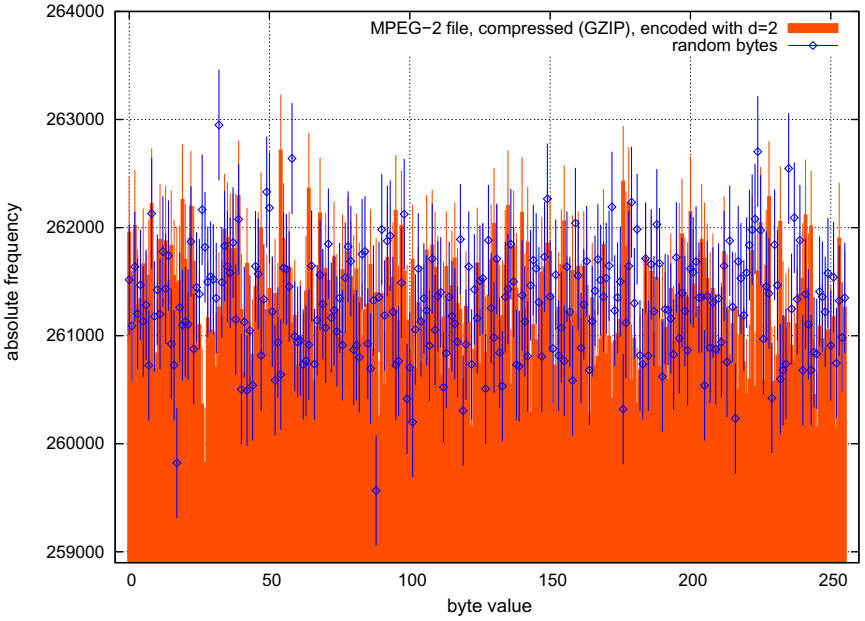

Fig. 12. Comparison of a pseudo-randomly generated file with a compressed (GZIP) MPEG-2 video stream purely encoded with $d = 2$.

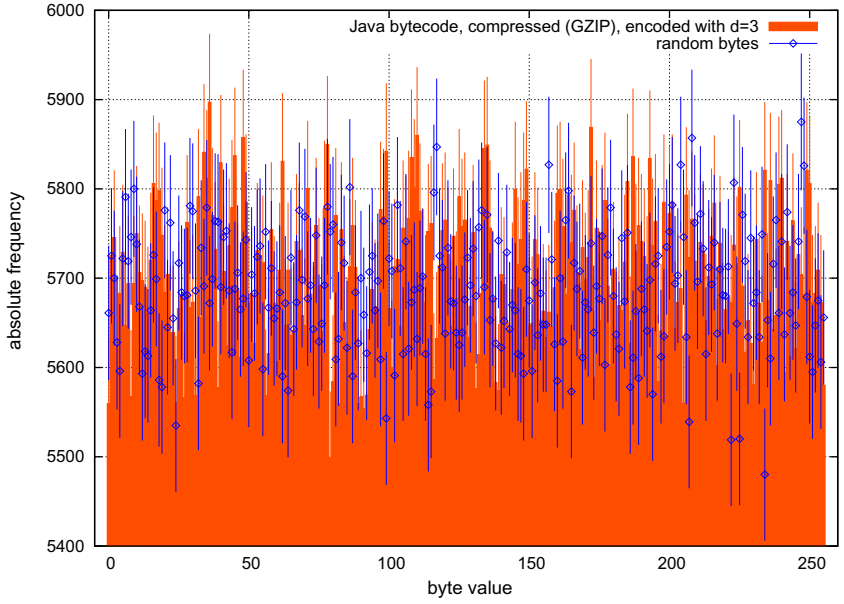

Fig. 13. Comparison of a pseudo-randomly generated file with compressed (GZIP) Java bytecode encoded with $d = 45$.

of two or more byte-blocks has not been considered. Theoretically the assembly of a file only consisting of purely randomly created bytes (if possible) should have no conspicuous structure at all. Therefore a file purely consisting of such pseudo-randomly generated data serves as our benchmark. If the output of the CNC has the same features as this benchmark, we assume that neither a prediction on the plaintext nor a prediction of the next encoded-byte is possible. The other testfiles are a videofile in MPEG-2 format and a file consisting of Java bytecode. The MPEG-2 file has been created with a low rate of lossy video compression and displays in a relatively high quality. The audio layer has been removed. The Java bytecode is archived in a JAR file and does not contain any Java documentations. The archive is not further compressed.

Our test environment contains two basic functions. The first function generates the histogram of the occurrences of a one-byte character set. Its input parameter is the testfile. To generate the histogram we use 256 counters to represent each state of one byte. The file is read one byte at a time. For every byte read, the referring counter is increased by one. Eventually each counter displays the absolute frequency of each state. To make a better distinction between an even frequency distribution and the random bytes, we use a second function to count the margins between the occurrences of an octet given as input symbol. The margin is represented by a counter, which contains the number of characters that appeared between the last and the next occurrence of the octet. Since we want to count all margins, the number of counters we use correlates with the absolute frequency of this byte, counted in the first

function. So we use one counter for each occurrence of this octet. To count the margins, the file is parsed one byte at a time. The goal of each step is to find the input character (octet). If the parsed byte does not equal this character, its referring counter is increased by one. Once the byte equals the input character the next counter is used. At the EoF each counter represents the actual number of characters between each occurrence of the input state. This is done for all states of one byte.

In a first approach we analyze the quantity of all possible states of one byte. Secondly we check the frequencies of such. Figs. 4 and 5 show the standard deviation from an equal distribution of each encoded packet. Figs. 6–13 show the histogram of the hole datastream considering all 256 characters representable by one byte. As shown in Fig. 6, a not further coded MPEG-2 file shows a relatively high structural occurrence compared to the referred random bytes. At this point the byte 0×00 is especially remarkable. Since it is one of the most relevant components in the structural information of an MPEG-2 file (sequence-start-code begins with 0×000001), we are not surprised to observe that it is the most frequently used character. Once fountain encoding is applied to the MPEG-2 file (here for presentation issues we purely encode all packets $\mathbf{x}$ with $d = 3$), these structures fade quite noticeably (see Fig. 7). Furthermore, as can be seen in Fig. 8, if the MPEG-2 file is encoded with $d = 4$, the resulting encoded packets $\mathbf{x}$ adapt almost completely to the structure of our referred pseudo-randomly generated file. All still noticeable peaks lie within the expected random fluctuation. Other formats show similar effects. However, the concrete results primarily depend on the characteristics of the underlying file format. As can be seen in Fig. 10 the result using Java bytecode encoded with $d = 8$ is not as promising. As depicted in Fig. 11 a degree of 45 is needed to achieve the same results as for the MPEG-2 video stream. The gap between the byte values 127 and 128 seen in Fig. 10 is a result of the initial frequency of the Java bytecode (see Fig. 9) in which mostly the byte values between 0 and 127 appear. This structure in the initial frequency distribution is a lot harder to obfuscate by randomly $\oplus$ plaintext packages.

If the testfile is now compressed before the encoding is applied the format dependency can almost be nullified. Most redundant information is reduced, thus decreasing the structural information. As seen in Figs. 12 and 13 we now get the same results with a degree of 2 for the MPEG-2 file and a degree of 3 for the Java bytecode. Since compression is a practical technique to make a statistical or cryptographic analysis more difficult, this approach can well be used to optimize the results of the obfuscation by CNC.

These effects can not only be observed by analyzing the whole datastream but also in each encoded packet. We now use function one for each encoded packet and calculate the standard deviation from each packet. In Fig. 4 the difference between an encoded packet with a degree of 2 and one with $d \geqslant 2$ is quite obvious. According to our preceding thesis of an WEP-like attack, this weak point could easily be exploited by a potential attacker. Once the datastream is compressed before encoding, this does not hold anymore. As seen in Fig. 5 it is now almost impossible to distinguish the degrees from one another.

We conclude that without using compression the success of the CNC approach is highly dependent on the underlying file format. Once the data is compressed this is no longer the case, which means that both in this section analyzed file formats are equally suited for data obfuscation with network coding.

9. Conclusion

Network coding is a viable technique to optimize the resources of a wireless network. We have shown that we can use the natural obfuscating effect of fountain codes to derive security means by purely encrypting the encoding vector, which is much smaller, hence saving computational costs. Thus, we do not encrypt the data itself, but we encrypt how it has been obfuscated. Our technique has the advantage to require very little overhead, and provides a better than nothing confidentiality mean. However, we acknowledge that if the confidentiality requirement is high or the source data too predictable, it is preferable to encrypt the complete source packets with more classical approaches. The proposed CNC solution is probably most suited in multi-hop settings with a binomial degree distribution at the initial sender side and in addition a full encryption of low-degree packets. For such a setting, in particular with an initial compression of the data, our evaluations on Java bytecode and MPEG-2 data streams have strengthened our confidence that CNC may be a reasonably good obfuscation mean to hide the content of a fountain encoded data stream.

Acknowledgement

The work presented in this paper was partly funded by the European Commission within the STREP WSAN4CIP project. The work of authors from HAW is partly funded by the German BMBF project SKIMS. The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies or endorsement of the WSAN4CIP project or the European Commission.

Appendix A. Impact on CNC of various network coding derivates

In our case study we analyze various network coding derivates to the limitation that CNC allows only random combination of encoded packets at intermediate nodes.

A.1. LT fountain code

To start the decoding process of some encoded packets on the fly it is essential that enoded packets with a low degree, e.g. $d = 1$ or $d = 2$ are transmitted. Generally we can say that $d_f \leqslant \sum d_i$ of $\mathbf{x}_f, \mathbf{c}_f$. Obviously the degree of the composed packet is $d_f = \sum d_i$ only in case the degrees of the received packets are not overlapping. We can state that in case the data stream is large and the degree is small we end up with a $d_f \leqslant \sum d_i$ with d_f near to the sum of all d_i. This is true for receivers at the first hop. However, in case the d_i of the incoming encoded packet is already large, e.g. $d_i \approx 1/2|\mathbf{c}|$ we state that the degree of the resulting encoded packet is also $d_f \approx \frac{|\mathbf{c}|}{2}$. Here the probability to have a even or odd number of '1'-bit per position in the various incoming coefficient vectors is 0.5, resulting in always $d_f \approx \frac{|\mathbf{c}|}{2}$. As we have seen previously in Fig. 1, random (since concealed) combination of incoming encoded packets $(\mathbf{x}_i, \mathbf{c}_i)$ due to *CNC* some few hops away from the source will almost surely result in a situation in which LT decoding is impossible. We conclude that CNC and the LT decoder is only compatible in single hop scenarios or if forwarder are not allowed to perform blind aggregation. Otherwise, a general decoder as explained in the next paragraph is needed.

A.2. Fountain code with binomial degree distribution

If the $\mathbf{c}_i$ are randomly chosen following a uniform distribution at the source, then the receiver can use a Gaussian elimination technique to decode when the received coefficient vectors achieve the full rank. Under such circumstances a binomial degree distribution has been proven to be advisable such that the effect for d_f to become approximately $\frac{|\mathbf{c}|}{2}$ is not a burden. However, the decoding complexity is $\Theta(n^3)$. For small $|\mathbf{c}|$, from a security perspective, hav-

ing a binomial degree distribution is good as it is resilient against attacker B, as described in Section 6. If $|\mathbf{c}|$ gets larger, the decoding burden becomes too high and the network coding becomes impractical. However, for large $|\mathbf{c}|$, tweaking the degree distribution with numerous low weight degrees provides both enough security, and a lower decoding overhead.

A.3. Raptor codes

The Raptor code is a two-tier coding approach with a combination of pre-encoding and LT channel coding. The main idea is that the receiver LT decoder does not need to recover every input symbol in order to retrieve the complete set of the source packets: The source generate δ check packets and uses the LT coding on the $n + \delta$ packets. This furthermore reduces the network overhead, while having a linear decoding complexity. However, the check packets generated by the pre-coding could help the attacker to attack with more ease the CNC scheme, assuming that the attacker knows about the pre-coding parameters.

References

[1] A. Hessler, T. Kakumaru, D. Westhoff, When Eco-IT meets security: concealed network coding for multicast traffic, in: IEEE International Conference on Pervasive Computing and Communications PerCom'10 – Sesoc Workshop, March 29–April 02, 2010.
[2] D. Dolev, A.C. Yao, On the security of public-key protocols, IEEE Transactions on Information Theory 29 (2) (1983) 198–208.
[3] Y. Fan, Y. Jiang, H. Zhu, J. Chen, X.S. Shen, Network coding based privacy preservation against traffic analysis in multi-hop wireless networks, <https://engine.lib.uwaterloo.ca/ojs-2.2/index.php/pptvt/release/view/8>, July 2009.
[4] N. Cai, W. Yeung, Secure network coding, IEEE International Symposium on Information Theory ISIT'02 (2002) 323.
[5] J. Feldman, T. Malkin, R.A. Servedio, C. Stein, On the capacity of secure network coding, in: 42nd Annual Allerton Conference on Communication, Control and Computing, September 2004.
[6] J. Tan, M. Medard, Secure network coding with a cost criterion, in: 4th International Symposium on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks, 2006, pp. 1–6.
[7] E. Tews, M. Beck, Practical attacks against WEP and WPA, in: The Second ACM Conference on Wireless Network Security ACM WiSec'09, March 2009, pp. 79–86.
[8] M.M. Hassanzadeh, M. Ravanbakhsh, O. Ytrehus, Two layer secure network coding – (2-LSNC), in: International Symposium on Telecommunications IST 2008, August 2008, pp. 7–12.
[9] L. Lima, M. Medard, J. Barros, Random linear network coding: a free cipher?, in: IEEE International Symposium on Information Theory ISIT'07, June 2007.
[10] S. Peter, D. Westhoff, C. Castelluccia, A Survey on the encryption of convergecast-traffic with in-network processing, IEEE Transactions on Dependable and Secure Computing 6 (January–March) (2010) 20–34.
[11] M. Krohn, M. Freeman, D. Mazieres, On-the-fly verification of rateless erase codes for efficient content distribution, in; IEEE Symposium on Security and Privacy, 2004.
[12] J.M. Bohli, D. Westhoff, A. Hessler, O.Ugus, Security enhanced multi-hop over the air re-programming with fountain codes, in: 4th IEEE International Workshop on Practical Issues in Building Sensor Network Applications SenseApp'09, October 2009.
[13] Z. Yu, Y. Wei, B. Ramkumar, G. Yong, An efficient signature-based scheme for securing network coding against pollution attacks, in: The 27th Conference on Computer Communications INFOCOM 2008, April 2008, pp. 1409–1417.
[14] S. Jaggi, M. Langberg, S. Katti, T. Ho, D. Katabi, M. Meard, Resilient network coding in the presence of byzantine adversaries, in: The 26th Conference on Computer Communications INFOCOM 2007, May 2007, pp. 616–624.
[15] T. Ho, B. Leong, R. Koetter, M. Medard, M. Effros, D.R. Karger, Byzantine modification detection in multicast networks with random network coding, IEEE Transactions on Information Theory, Special Issue on Information Theoretic Security 54 (6) (2008).
[16] W. Lou, W. Liu, Y. Fang, SPREAD: Enhancing data confidentiality in mobile ad hoc networks, in: The 23th Conference on Computer Communications INFOCOM 2004, vol. 4, May 2004, pp. 2404–2413.
[17] C.K.L. Lee, X.H. Lin, Y.K. Kwok, A multipath ad-hoc routing approach to combat wireless link insecurity, in: IEEE International Conference on Communications ICC'03, vol. 1, May 2003, pp.448–452.
[18] C. Castelluccia, E. Mykletun, G. Tsudik, Efficient aggregation of encrypted data in wireless sensor networks, in: The Second Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services Mobiquitous, July 2005, pp. 109–117.
[19] T. ElGamal, A public-key cryptosystem and a signature scheme based on discrete logarithms, in: Annual International Cryptology Conference CRYPTO'85, vol. IT-31, no. 4, July 1985, pp. 469–472.
[20] M. Luby, LT codes The 43rd Annual IEEE Symposium on In Foundations of Computer Science, November 2002, pp. 271–280.
[21] J. Feldman, N. Yakovenko, J. Cranshaw, Rateless Codes, Scribe notes, COMS W4995, Introduction to Coding Theory, Columbia University, New-York, 2004. pp. 18–22.

Please cite this article in press as: A. Hessler et al., Data obfuscation with network coding, Comput. Commun. (2010), doi:10.1016/j.comcom.2010.11.004