# XOR Network Coding Pollution Prevention Without Homomorphic Functions

Juan Camilo Corena*†, Anirban Basu*, Shinsaku Kiyomoto*, Yutaka Miyake*, Tomoaki Ohtsuki†
*KDDI R&D Laboratories Inc. 2-1-15, Ohara, Fujimino-shi, Saitama 356-8502, Japan
†Graduate School of Science and Technology, Keio University 3-14-1 Hiyoshi, Kohoku-ku, Yokohama, 223-8522, Japan

***Abstract*—Network coding is a way of transmitting information where nodes in a network combine incoming packets into a single one to increase throughput in some scenarios, nodes wishing to get the original information can perform decoding when enough packets have been received. Given its efficiency, the exclusive or (XOR) operation is very popular for network coding. One security concern for networks using network coding is the so called "pollution attack", where an adversary introduces packets that are not combinations of the original ones. In this paper, we present a construction to prevent pollution attacks in XOR network coding that is suitable for networks where nodes must perform fast verifications. Unlike existing constructions in the literature which are based on XOR-homomorphic authentication functions, our construction can be instantiated with existing cryptographic primitives that are not related to the XOR operation. The core insight of our proposal is a carefully selected set of authenticated packets that are used to authenticate the network coding stream. We show that our proposal is computationally efficient at the intermediate nodes and that can be computed efficiently at the nodes which are generating the content.**

## I. Introduction

Network coding is a useful technique to increase throughput in networks, which can be used in any network where there is either, more than one path from transmitter to receiver, or several receivers. Such conditions are usually present in several M2M(Machine-to-Machine) communication scenarios, such as: smart grids, home-network appliance networks, and sensor networks. Consider a scenario where a power company wishes to upgrade the information related to energy costs stored at the smart meters located at each household [1] using a wireless network, where the meters act as intermediate relays for the communication. In this scenario a network coding strategy could be used to deliver information faster by reducing the number of retransmissions needed due to packet loss.

Despite its advantages, using network coding introduces new types of attacks that can overturn network coding benefits. Because of this, lightweight security primitives for network coding are necessary. The challenge in creating digital signatures or messsage authentication codes (MACs), which can be used for network coding, is the number of possible messages that can be produced as a result of combining the original information packets. Because of this significant number of possible combinations, the usual strategy for linear network coding involves signing a base of the space spanned by the packets and then using a function with linear properties, to infer the right signature value for a particular combination.

Security properties of these functions, must guarantee that it is hard for an adversary to find a valid signature for elements outside the space spanned by the packets, but easy for anyone to find the signature for elements in the space, once the signature for the basis has been seen. This has been done in the literature using either the linear properties of discrete logarithms [2], bilinear pairings [3], dot products among random vectors [4], or vector orthogonality [5]. Security for the systems is usually associated with the size of the finite field used for the operations.

### A. The problem

Unlike the linear case, XOR network coding uses a very small finite field for computations and cannot take advantage of the previous constructions. Despite this, some constructions addressing the problem have been proposed such as [6], which uses XOR-homomorphic Message Authentication Codes (MACs). This means that for messages $m_1, m_2$ and their respective authentication codes $\sigma(m_1), \sigma(m_2)$, the following holds:

$$\sigma(m_1 \oplus m_2) = \sigma(m_1) \oplus \sigma(m_2),$$

where $\oplus$ is the binary XOR operation. Then, by using a key assignment, intermediate nodes can check the packet is not polluted. More general approaches, such as the one presented in [7] based on ideal lattices, can be used for small fields to authenticate any polynomial function including network coding.

### B. Our construction

Our construction takes an entirely different approach and does not rely on any homomorphic properties of the authentication function. The intuition for our construction is based on one key point: coding strategies for XOR Network Coding are different from those of Network Coding in larger finite fields. Unlike traditional approaches that wish to authenticate all possible linear combinations, we only care about the combinations that are likely to occur given the coding strategy used in the network.

Contributions of this paper come from several aspects. To the best of our knowledge, we are the first to provide a solution to several XOR network coding scenarios without homomorphic functions, which can be slow, require key management scheme for the MAC variants [4] or require additional assumptions on the system such as time synchronization [8]. Our proposal had fast verification times for all scenarios once packets could be decoded. For scenarios where immediate decoding was not possible, we proposed a way to include packets in the accumulator that guarantees at worst logarithmic

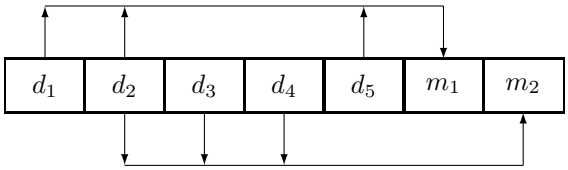



Figure 1. MAC computation from [6]. Random positions of the payload $d_1, d_2, d_3, d_4$ are selected for a particular MAC $m_1, m_2$, selected positions are XORed together and then XORed with a value generated using a secret key.

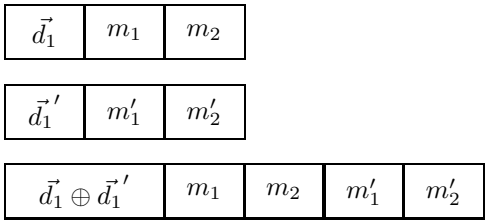



Figure 2. Combination step from [6], where two packets are combined into a single one. The original payloads $\vec{d_1}$ and $\vec{d_1}'$ are XORed, whereas all the MACs from the original packets are appended to the resulting one.

retransmision overhead (in the size of the generation) and in average 1 packet retransmission overhead for our target generation of size 16, while reducing the number of packets that must be added to the accumulator by a factor of 4. In addition, our proposal can be instantiated using off-the-shelf cryptographic functions, which makes its deployment for real systems easier.

The rest of the article is organized as follows: in section II previous work in pollution prevention for XOR network coding is presented; in section III, building blocks needed for our proposal are introduced; in section IV we present the 3 variants for our proposal, depending on the size of the generation and the type of accumulator used; in section V we show the results of our simulation experiments; finally section VI presents the conclusions of this work.

## II. Related work on pollution prevention for XOR network coding

In [6], a MAC-Based system was proposed. The idea of their construction is to use random elements of the packet's payload to create a single MAC. The process is shown in Fig. 1, where the payload is represented by $d_1, d_2, d_3, d_4$ and the computed MACs by $m_1, m_2$, the arrows indicate what positions of the payload were used to create a particular MAC. In particular, $m_1$ was formed using $d_1, d_2$ and $d_5$. The actual computation involves XORing the involved positions and then XORing a random value generated by a secret key. The process of combining two packets, involves XORing the payloads of the two packets together and appending the MACs of both packets to the resulting one. The process is shown in Fig. 2, where the payloads are represented by vectors $\vec{d_1}, \vec{d_2}$. As a result of this procedure, the packets increase in length depending on how many of them have been combined so far. To verify a packet, the generation routine is repeated and compared to the result of the XOR of the selected positions in the received packet.

In [4], a homomorphic MAC construction that can be used for XOR network coding was proposed. Its construction is as follows: create a network coding matrix, each element inside the matrix will be an element of $\mathbb{F}_2$, this matrix has the form $[\boldsymbol{ID}]$, where $\boldsymbol{I}$ is the identity matrix and $\boldsymbol{D}$ is the original message to be transmitted in matrix form. A random vector from $\vec{v} = \mathbb{F}_2$ is generated using a private key and $[\boldsymbol{ID}] \cdot \vec{v}$ is computed; a second key is used to generate a second vector $\vec{w} \in \mathbb{F}_2$. A homomorphic MAC for a vector formed as a linear combination of the rows of $[\boldsymbol{ID}]$ is given by

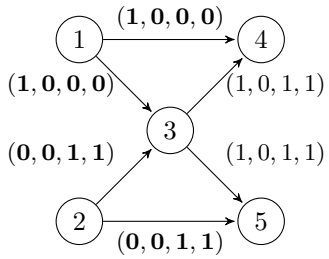
$$\sum_{r \in \text{Rows}([\boldsymbol{ID}])} v_r \oplus w_r \tag{1}$$

The output of the MAC is an element of $\mathbb{F}_2$. This MAC can be used on larger finite fields and the probability of an adversary guessing the right value for a given packet, is given by $1/|\mathbb{F}|$, where $|\mathbb{F}|$ is the size of the finite field. Since we are interested in $\mathbb{F}_2$ vectors, an adversary can guess the right value with probability $1/2$, for this reason several MACs need to be computed to achieve enough security. Packets can be XORed together and nodes in possession of the keys can verify it is not polluted. If internal attackers are considered, even more keys are needed to protect the system from keys known to the adversary. As in any MAC-based system, the key assignment strategy is important to create a trade-off between security and information overhead. Unlike the previous scheme, packets do not increase their length as more XOR operations are performed. In [8], the authors proposed a similar system. Like the previous construction, a random vector is XORed to the information to be transmitted. The difference is that by using loose time synchronization, the secret value used to creat one MAC is sent after the message has been transmitted. This secret value is authenticated using a fast mechanism. Since the secret is unknown when the packets were transmitted, receivers can be sure the message is authentic.

## III. Preliminaries

### A. XOR network coding routing protocols

Unlike linear network coding over larger fields, where each packet is a linear combination of all the packets to be transmitted, XOR network coding takes a different approach based on coding opportunities. Informally speaking, a coding opportunity arises when a node can deliver information simultaneously to more than one node by transmitting a single packet. In Fig. 3, there is a coding opportunity for node 3, because node 4 wants a packet node 5 has, while node 5 has a packet node 4 wants. Therefore, by computing the XOR of the two packets, a packet that provides new information to both nodes simultaneously is produced. A protocol wishing to take advantage of these opportunities must include a way for nodes to transmit the set of packets each node "has" and "wants". Since nodes receiving encoded packets can decode immediately, this type of network coding is known as Immediately Decodable Network Coding (IDNC).




| Authenticated Packets | |
|---|---|
| $(0,1,0,0)$ | $(0,0,0,1)$ |
| $(\mathbf{1},\mathbf{0},\mathbf{0},\mathbf{0})$ | $(0,0,1,0)$ |
| $(1,1,0,0)$ | $(\mathbf{0},\mathbf{0},\mathbf{1},\mathbf{1})$ |

| Non-Authenticated Packets |
|---|
| $(1,0,1,1) = (\mathbf{1},\mathbf{0},\mathbf{0},\mathbf{0}) \oplus (\mathbf{0},\mathbf{0},\mathbf{1},\mathbf{1})$ |

Figure 3. A XOR-network-coding network with a generation of size $n = 4$. There is a set of authenticated packets of size 6, which can be used to authenticate packets that are not in the list, by performing XOR ($\oplus$) elimination on the received packets. For instance: Node 4 can verify $(1,0,1,1)$ by verifying $(1,0,0,0)$ is in the list of authenticated packets and then computing $(0,0,1,1) = (1,0,1,1) \oplus (1,0,0,0)$; now, the result can be found in the list of authenticated packets.

In mathematical terms, a coding opportunity can be defined using a graph $G$. Vertices of $G$ are labeled as $v_{i,j}$ where $i$ is a node in the network and $j$ is a packet of the current generation. A vertex is created only when node $i$ wants packet $j$ [9]. For instance: before transmitting any packets, the graph corresponing to Fig. 3 for a generation of size $8$, would have $40 = 5 \cdot 8$ vertices $v_{1,1}, \ldots, v_{1,8}, \ldots, v_{5,1}, \ldots, v_{5,8}$. Two vertices $v_{i,j}, v_{k,l} \in G$ are connected when any of the following situations occur:

1) If $j = l$: this means node $i$ and node $k$ want the same packet.
2) If node $i$ wants packet $l$ and node $k$ wants packet $j$: one node has a packet the other wants and viceversa.

In the previous graph, every subgraph where all vertices are connected (clique) is a coding opportunity. After finding a clique in the graph, all the packets involved in the set of vertices forming the clique are XORed together. This construction guarantees that all the nodes which appear in the clique can decode the packet immediately after reception. In terms of overall network performance, it is desirable to select the cliques with the largest number of nodes, because that benefits the largest number of nodes. By analizing the properties of the cliques of this graph, it is possible for the source to select packet combinations that are likely to occur during transmission.

Even though finding cliques in graphs is an NP-Hard problem in the general case, for the context of XOR network coding, it is possible to find cliques that improve network throughput using strategies such as: retransmitting the packet wanted by most nodes [10]; selecting a packet wanted by at least one node at random, then combining it with other packets as long as the resulting packet can be decoded immediately by at least a given number of neighbors on the next hop [11]; or exhaustive search with smart prunning [9]. Traditional graph algorithms such as [12] can also be used.

In section section V-C, we present a general strategy to include elements in the accumulator, such that any possible packet in a generation, can be represented with $\mathcal{O}(\log(g))$ elements that are contained in the accumulator. Our strategy works independently of routing strategy but can be combined with any of the previously presented coding methods, to reduce the number of packets that must be added to the accumulator, while providing enough coding options for relays.

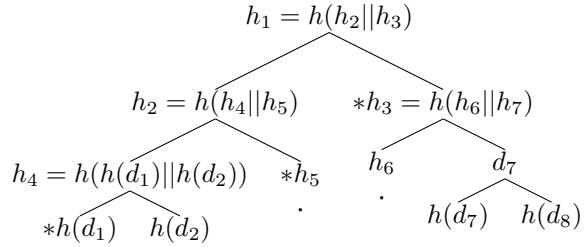



Figure 4. Accumulator based on a Merkle Tree for values $h_1 \ldots, h_8$. Leaves of the tree are the output of a hash function of the original data, internal nodes of the tree are computed as the hash of the concatenation of its children. To prove a value belongs to the accumulator, the siblings of the path from the node to the root must be included with the packet. The proof for $h_2$ is given by members marked with an "$*$".

### B. Cryptographic Accumulators

Cryptographic accumulators were first proposed in [13]. As explained in [14]: *an accumulator scheme is an algorithm to combine a large set of values into one, short accumulator, such that there is a short witness that a given value was indeed incorporated into the accumulator*. In this section, we will introduce two accumulator constructions, one based on a Merkle Tree [15] and another one based on a Bloom Filter [16]. The relevance to our construction is that in one of them, the witness is the message itself, whereas in the other, additional information must be provided. Our proposal can be instantiated with any accumulator as long as it is hard for an adversary to forge membership of elements.

The first of the accumulators is based on a Merkle Tree, which is a full binary tree where the leaves are the hashes of the elements that are included in the accumulator. Internal nodes of the tree are computed as the hash of the concatenation of the children of that node. The root of the tree is authenticated and transmitted to nodes in the network. The hash function $h$ used for this accumulator must be hard to invert (preimage resistance). To prove an element is in the accumulator, the element, along with the siblings of its path to the root are transmitted. Using these values it is possible to reconstruct the path all the way to the root of the tree. Security properties of the system follow from the preimage resistance of the hash function, since it is unfeasible for an attacker to create a valid path to the root, without inverting the hash function. The process is illustrated in Fig. 4 where it is shown what information must be sent to prove the membership of $d_2$; for this case $d_2$ and the siblings of the ancestors of $h(d_2)$ can be used to reconstruct the path to the root.

An approach where only the message is necessary to prove membership, was presented in [17]. To add an element $y$ to the accumulator, one computes $\overline{y} = h(y)$, here it is assumed that $\overline{y}$ is $rd$ bits in size. Next, $\overline{y}$ is treated as $r$ different $d$-bit numbers $(\overline{y}_1, \ldots, \overline{y}_r)$, for each $\overline{y}_i$ a single bit $b_i$ is output, where $b_i = 1$ if $\overline{y}_i \neq 0$ and 0 otherwise. The value for the

accumulator is the binary AND operation of the respective bit vectors computed from the elements. Unlike the Merkle-tree-based approach which needs an additional witness, only the message is needed.

Cryptographic accumulators in network protocols, have been used in a construction called *Distillation Codes* [18], which was used for multicast authentication. In that construction, information to be transmitted is cryptographically authenticated and then encoded using an erasure code, the output of the erasure code is added to the accumulator. Then, the encoded pieces are sent along with a proof of membership to a given accumulator. Receivers can associate encoded pieces to different original packets, since it is impossible for an adversary to create an encoded piece that passes the accumulator test. Once enough encoded pieces have been received from a particular accumulator, the decoding procedure is invoked; if the decoded information produces information that can be authenticated, the data is accepted; or discarded otherwise. Our construction can be seen as an extension of this construction, since the original did not consider scenarios where encoded pieces could be further encoded by intermediate nodes.

## IV. Proposal

As pointed out in the introduction, our idea consists in creating a trade-off between the size of the list of authenticated packets and performing decoding operations. In more detail, the scenario depicted in Fig. 3, shows a classical butterfly topology for a generation of size $g = 4$. The binary values transmitted among the different nodes represent what packets were XORed to produce the given packets; to keep the example simple, the packets contain no payload. If we apply the straightforward approach of signing all the possible combinations for a generation, it would be necessary to sign $2^4 = 16$ packets; however, the list of authenticated packets $L$ which is known to all nodes, only contains $6$ elements. In the figure, node $3$ performs the XOR of the two incoming packets $(1,0,0,0), (0,0,1,1) \in L$, to compute $(1,0,1,1)$ and transmit it to nodes $4, 5$. Given that packet $(1,0,1,1) \notin L$, nodes $4, 5$ do not know whether it is polluted or not. Fortunately, node $4$ received $(1,0,0,0)$ from node $1$ and node $5$ received $(0,0,1,1)$ from node $2$. If nodes $4, 5$ apply their received authenticated packets to the newly received one, the result is a packet in $L$.

This approach is known in the literature as "signature amortization" [19], because it amortizes the cost of verifying the signature among all the packets contained in the list, since the signature is only verified once. At first glance, the system might seem inefficient, given that we need to sign $2^g$ packets. However, this is not necessarily the case as shown in Fig. 3, where decoding steps are performed to reduce the size of the list of authenticated packets. The actual implementation of our idea does not rely on a list but on the use of a cryptographic accumulator, which is a compact way to verify the packet is on the list without transmitting the list.

In cases where it is not possible to reduce the packet to one that belongs to the list, such as the one depicted in Fig. 5, we propose a method to create the list and also that makes the best use of the packets available in the list. We prove that when our method is used to represent any packet as a combination

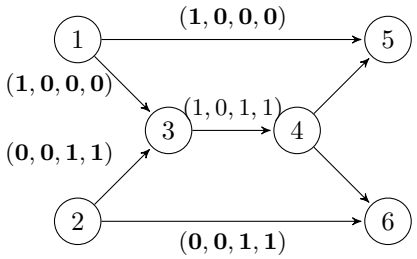



Figure 5. An example without instant decodability. Node $4$ cannot forward packet $(1,0,1,1)$ because it is not in the authenticated list of packets. The list of authenticated packets is the same as Fig. 3

of packets that can be authenticated, the overhead in terms of retransmissions is never larger than the logarithm of the generation size.

In this section, we describe $3$ variants which depend on the size of the generation involved, the number of packets that are XORed together (degree) and the coding strategy used by nodes in the network.

### A. Exhaustive inclusion strategy

For a XOR network coding network, given a generation of size $g$, there are $2^g$ possible packet combinations. For small values of $g$, it is possible for a source to compute all the possible XOR combinations among the packets and add them to a cryptographic accumulator.

**Initialization:** Add all the possible packet combinations to the cryptographic accumulator.

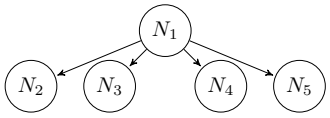
**Encoding:** Encode packets according to some scheme. The details of the coding scheme are not relevant this variant of the system since all possible combinations are included in the accumulator.

**Verification:** Verify every incoming packet is in the cryptographic accumulator; discard otherwise.

To make the computation more efficient, XORs of the different combinations can be computed using the binary Gray Code. In this way, it is only necessary to compute the XOR of one packet per round, before adding the value to the authenticator. Authenticators needed for this construction must not have a proof of membership besides the message itself, otherwise packets cannot be authenticated. Nyberg's authenticator explained in Section III-B has this property.

### B. Inclusion strategy for Immediately Decodable Network Coding

The drawbacks of the previous construction are computational overhead at the source and a constraint on the generation size. These drawbacks can be overcome by adding a constraint on the coding strategy of the nodes. A popular strategy for XOR network coding called Instantly Decodable Network Coding (IDNC) [11], guarantees that packets are decodable at the next hop (in Fig. 3 this is the case). This guarantees that after one hop, nodes can use the accumulator to efficiently verify the decoded packet was transmitted by the source, for this reason only the original packets must be included in the




| Initital State | |
|---|---|
| $N_1$: $A, B, C, D$ | |
| $N_2$: $A$ | $N_3$: $B$ |
| $N_4$: $C$ | $N_5$: $D$ |

Figure 6. Sample network topology with nodes $N_1, N_2, N_3, N_4, N_5$ and packets $A, B, C, D$. The state shows what packets have been received by each node.

accumulator. The information to select the right packets to achieve this property comes from feedback sent by the nodes; using this information, transmitters can select a set of packets to code that satisfies the needs of the receivers.

**Initialization:** Add packets of degree 1 to the cryptographic accumulator.

**Encoding:** Encode maximizing the nodes that can decode packets immediately. This can be done by using an algorithm such as the BronKerbosch algorithm [12] to find cliques in the graph representing the coding opportunities, or specialized algorithms such as the one presented by Le et al. in [9].

**Verification:** Reduce the degree of the incoming packet and verify it is in the accumulator; discard otherwise. Given the IDNC coding strategy, this process is performed by reducing the degree of the incoming packet using authenticated degree-1 packets. Then, verify the resulting packet is in the accumulator.

This construction is similar to the exhaustive inclusion one; the difference is that only packets of degree one are present in the accumulator. For this reason, it is necessary that the coding strategy limits itself to transmit packets that can be immediately decoded by the receivers. The main advantage of this construction compared to the previous one, is that the size of the generation can be much larger. This also implies that computation at the source can be significantly smaller. Regarding the selection of the cryptographic authenticator, in this case it is possible to use authenticators with a separate proof of membership, such as the Merkle Tree construction. This is possible since the accumulator only contains packets of degree one.

### C. General inclusion strategy

Despite its efficiency and popularity, IDNC does not necessarily offer optimal throughput. Consider the scenario depicted in Fig. 6 and the sequence of packets transmitted by $N_1$ shown on Table I. On the left table, strickenout packets are those that have been received by nodes, but that cannot be decoded immediately; since IDNC does not consider buffering these packets, they are discarded. This is the case for packet ~~$A \oplus C$~~ which cannot be decoded by $N_3$ and $N_5$. In the general case, non-decodable packets can be used later to improve throughput, as it can be seen in the right part of Table I, where only 3 packets are needed as opposed to 4 in the IDNC scenario.

The steps for this scenario can be summarized as follows:

**Initialization:** Add combinations that are likely to occur in the accumulator.

Table I. DIFFERENCE BETWEEN IMMEDIATE DECODING AND THE GENERAL CASE FROM DECODING PERSPECTIVE

| Immediately Decodable Case |
|---|
| $N_1$ sends $A \oplus C$ |
| $N_1$: $A, B, C, D$<br>$N_2, N_4$ : $A, C$<br>$N_3$ : $B$,~~$A \oplus C$~~<br>$N_5$ : $D$,~~$A \oplus C$~~ |
| $N_1$ sends $B \oplus D$ |
| $N_1$: $A, B, C, D$<br>$N_2, N_4$: $A, C$, ~~$B \oplus D$~~<br>$N_3, N_5$: $B, D$ |
| $N_1$ sends $D \oplus C$ |
| $N_1$: $A, B, C, D$<br>$N_2, N_4$: $A, C, D$<br>$N_3, N_5$: $B, C, D$ |
| $N_1$ sends $A \oplus B$ |
| $N_1, N_2, N_3, N_4, N_5$: $A, B, C, D$ |

| General Case | |
|---|---|
| $N_1$ sends $A \oplus C$ | |
| $N_1$: $A, B, C, D$ | |
| $N_2$: $A, C$ | $N_3$: $B$, $A \oplus C$ |
| $N_4$: $A, C$ | $N_5$: $D$, $A \oplus C$ |
| $N_1$ sends $B \oplus D$ | |
| $N_1$: $A, B, C, D$ | |
| $N_2, N_4$: $A, C$, $B \oplus D$ | |
| $N_3, N_5$: $B, D$, $A \oplus C$ | |
| $N_1$ sends $D \oplus C$ | |
| $N_1, N_2, N_3, N_4, N_5$: $A, B, C, D$ | |

**Encoding:** Encode maximizing throughput according to some strategy.

**Verification:** Verify every incoming packet according to Algorithm 1.

In this scenario, finding packet combinations that are likely to occur in the system is not as easy as in the IDNC scenario, because they depend on the topology of the network, packet loss rate, interference, noise and other factors. Since we are concerned with generic strategies that can work for any network, we suggest adding packets to the accumulator whose degree is a power of two. The motivations for this strategy is that without knowledge of the current system, it guarantees that any packet outside the accumulator can be produced using a logarithmic number of packets (in the degree of the packet) that can be authenticated. This is useful in cases when the best packet for a particular situation is not contained in the accumulator. In that case it is guaranteed that a particular packet can be replaced by a not too large number of packets that can be authenticated. We formalize the previous claim in Theorem 1.

**Theorem 1.** *Let $p$ be a packet that must be sent to neighbor nodes $n_1, \ldots, n_\eta$. Let $P = \{p_1, \ldots, p_\delta\}$, be a set of packets such that $\bigoplus_{i=1}^{\delta} p_i = p$ where the degre of all $p_i = 1$. Let $g$ be the size of a network coding generation and $A$ be an accumulator containing all packet combinations where the degree of the combination is a power of two. Then, $p$ can be transmitted using $\mathcal{O}(\log(\delta))$ packets by performing combinations of elements in $P$.*

*Proof*: Since all packets $p_i \in P$ have degree 1 and are different by the definition of a set; then, $P \oplus p_i$ has degree $\delta - 1$ and $p_i \oplus p_j = 2$ for $i \neq j$. Now, select without replacement, $c$ elements from $P$, where $c$ is the largest power of two which is less than or equal to $\delta$ and apply the XOR of the selected packets to $p$.

$$p' \leftarrow p \oplus (p_1 \oplus \ldots \oplus p_c) \tag{2}$$

here, $(p_1 \oplus \ldots \oplus p_c)$ is the packet to be transmitted.

Since the degree of $(p_1 \oplus \ldots \oplus p_c)$ equals $c$, because packets are different; then, the degree of $p'$ is $\delta' = \delta - c$. The process is iterated until the degree of the remaining packet is 0. Given that in each iteration $c \geq \delta/2$, then $2\delta' \leq \delta$. This shows that the degree of the original packet can be reduced by at least

**Algorithm 1** General Pollution Detection Algorithm

```
1: function VERIFY(p, T, V, L, τ)
2:     T ← T ∪ {p}
3:     if |T| > τ then
4:         Remove a random element from T
5:     end if
6:     repeat
7:         changed ← FALSE
8:         for each t ∈ copy(T) do
9:             Remove t from T
10:            B ← Reduce(V,t,L)
11:            if B is empty then
12:                Add t to T
13:            end if
14:            for each b ∈ B do
15:                if b ∈ Accumulator then
16:                    Add b to V
17:                    changed ← TRUE
18:                else
19:                    break
20:                end if
21:            end for
22:        end for
23:    until changed = FALSE
24: end function
```

half on each iteration, hence the algorithm needs $\mathcal{O}(\log(\delta))$ subsets of $P$ to produce the original packet $p$.

What remains is to show that the packets needed to represent $p$ are in the accumulator. By the definition of accumulator $A$, packets whose degree is a power of two are included; therefore the packets found by the algorithm belong to the accumulator. □

The final step, involves checking that incoming packets are not polluted, this part is handled by Algorithm 1. Algorithm 1 first starts by adding the received packet $p$ into the set of temporary packets $T$ which have not been verified for pollution. When the number of buffered packets in $T$ exceeds the available memory, one packet is evicted randomly.

The next step involves reducing the degree of the packets in $T$, using elements from the set of already verified packets $V$. This could be done using Gaussian Elimination (GE) in $\mathbb{F}_2$; by reduction we do not necessarily mean decrease the actual Hamming weight of the coefficients vector, but rather use it to find vectors belonging to the accumulator. What "Reduce" does, is to perform row reduction on the matrix $[v_i \in V, t, l_i \in L]^{'}$ using the first $|V|+1$ rows as pivot rows which we represent using $RR_{|V|+1}([v_i \in V, t, l_i \in L]^{'})$, here "$'$" represents the transpose operator. Then, it should find what members of $l_i$ became the $[0]$ vector, which means these authenticated packets can be created as a combination of the existing ones. Since this process is computed several times, some members of $l_i$ will be $[0]$ after they can be created for the first time; in that case, we only care about the first time this happens for each vector $l_i$. It is important to mention that just discovering a new packet, does not mean the packet provides new information; for a packet to actually provide new information, it must be linearly independent from the previous members of $V$. The previous insights are contained in Algorithm 2, where the matrix is reduced, then new vectors in the accumulator are selected and stored in $Z$, and finally a basis $B$ of $Z$ is selected as the newly decoded useful packets that can be tested in the accumulator.

**Algorithm 2** General Pollution Detection Algorithm

```
1: function REDUCE(V, t, L)
2:     M ← RR_{|V|+1}([v_i ∈ V, t, l_i ∈ L]')
3:     Z ← vectors l_i ∈ L that became [0] for the first time
4:     B ← a basis for the linear span of Z
5:     return B
6: end function
```

### D. Security Analysis

From a theoretical perspective, the three variants of our proposal guarantee that packets are never combined with other packets, until their memberships have been tested in the accumulator. For this reason, the protocol is as secure as the accumulator used. From an implementation perspective, in the three variants, nodes must verify the membership of incoming packets. This could be used by an attacker wishing to exhaust nodes' resources, hence accumulators with low computational complexity must be selected. In particular the accumulators presented in Section III-B, were selected for explanation purposes, because they are compuationally light. Despite this, Nyberg's accumulator [17] must be used with utmost care in practice, since it has a high false positive rate if parameters are not selected correctly.

An additional security concern is present for the general inclusion strategy, given that packets are evicted from the queue of unverified packets whenever the queue is full. An attacker that is able to inject a significant number of packets may cause eviction of valid packets. To deal with this attack, an aditional reputation mechanism could be used to prioritize eviction of nodes who do not provide enough valid packets. To prevent abuse of this measure, it should be enforced with an identification mechanism.

## V. SIMULATION RESULTS

Simulations were performed on a Windows 8 PC, with 16GB in main memory and an Intel Core i7 − 3770 CPU running at 3.4GHz with an 8MB cache. The implementation of the cryptographic primitives was made using the Java JDK 1.7 update 21. Unless otherwise stated, the execution times presented in this section, are the average of running the simulation 1000 times.

### A. Exhaustive inclusion strategy

We performed several simulations to measure the efficiency of the proposal, the first of them is shown in Fig. II. The table shows execution times needed to compute all the possible packet combinations of a given generation as well as the variants that only add packets of degree $2^k$; for the columns using hash functions, a Nyberg accumulator was computed. To generate enough cryptographic material the hash function was invoked several times, taking as input the previous output.

Table II. EXECUTION TIMES FOR CREATION USING THE EXHAUSTIVE INCLUSION WITH 1500 BYTE PACKETS

| Gen. size | MD5 | MD5-$2^k$ | SHA | SHA-$2^k$ | No Hash |
|---|---|---|---|---|---|
| 4 | 0.18 ms | 0.10 ms | 0.15 ms | 0.10 ms | 0.02 ms |
| 8 | 1.14 ms | 0.47 ms | 1.69 ms | 0.70 ms | 0.2 ms |
| 12 | 18.13 ms | 4.76 ms | 26.59 ms | 7.05 ms | 3.08 ms |
| 16 | 285.95 ms | 69.68 ms | 425.17 ms | 100.78 ms | 49.16 ms |

The simulation wanted to verify the overhead induced by the selection of the hash function in our proposal; the hash functions that were tested were MD5 [20] and SHA-1 [21] due to their availability in the Java JDK, however MD5 must not be used for a real implementation since it is no longer secure. The final column shows the execution times of just creating all the XOR combinations for the packets in the generation, without creating the accumulator.

As it can be seen from table results, the selection for the hash function greatly impacts the ability of the source to create packets. This implementation used a single thread and a Grey Code approach to XOR a single packet per hash function invocation when all possible combinations were added to the accumulator. A performance improvement is seen for the $2^k$ variants of the algorithms, since less number of combinations are computed, time is reduced, the cost that is payed in this case is that not all possible packets can be authenticated immediately, which influences the coding strategy of nodes. Even though the procedure could be performed in parallel, the exponential growth in the number of combinations confines this proposal to small generation sizes. This proposal is suited for information that is known in advance, such as software updates; for information generated in real time, it is possible to use it depending on the throughput needed. For instance, the SHA-$2^k$ implementation shown on the fifth column of Table II, could be used to transmit at approximately $232.5$ KBps, a parallel implemenation could scale this rate linearly and the selection of a faster hash function can increase throughput as well to match application needs.

Adding only packets whose degree is a power of $2$ to the accumulator increases the number of packets that must be transmitted. For the case of a generation of size $16$ on average $2.03$ packets are needed in the $2^k$ version, compared to the optimal case where all packets are included in the accumulator. This value was found by exhaustively checking all the possible coefficient combinations for each generation; the value for a generation of size $32$ was found to be $2.48$. This value is consistent with Theorem 1 which gives an upper bound for a single packet. To put this results into context, assume that in a network not using network coding, in order to satisfy the needs of a group of nodes, $n$ packets must be retransmitted. Let us say that by using XOR Network Coding, this number can be reduced to $r < n$ packets. What the previous result tells us, is that for a system using the $2^k$ strategy for $g = 16$, in average the transmitter will send $2.03 \cdot r$ packets that can be immediately authenticated to satisfy the needs of the receivers. Given the bound from Theorem 1, the transmitter will never transmit more than $4 \cdot r$ packets in that scenario.

Table III. EXECUTION TIMES FOR CREATION USING THE IDNC VARIANT WITH 1500 BYTE PACKETS

| Generation size | MD5 ($\mu$s) | SHA ($\mu$s) |
|---|---|---|
| 8 | 36.36 | 50.14 |
| 16 | 59.47 | 93.58 |
| 32 | 119.29 | 187.51 |
| 64 | 238.96 | 382.15 |

### B. IDNC

The next simulation involved the IDNC scenario, results are summarized in Table III. Since this variant involves no XOR operations, the "No hash" column is not included. This variant is significantly faster than the small-generation one, since only one hash invocation is needed per packet. Results show that our implementation is in the order of $\mu$s instead of ms as in the previous scenario. For this reason, this construction can be used with real time content, for which IDNC is a good technique to improve throughput. For this variant a Merkle Tree accumulator was used.

### C. General inclusion strategy

For the general case, testing whether an unverified packet generates a packet in the accumulator for a generation of size $16$, takes approximately $47$ $\mu$s in our Gaussian Elimination routine using bitwise, for this reason the buffer of temporary packets must be kept small. For this experiment the number of possible packets in the accumulator was set to $14827$ since that is the number of combinations added when only packets with degrees that are a power of two are added. To reduce the processing time, it is recommended that a sender willing to use packets not in the accumulator, should inform receivers of what packets in the accumulator are expected to be decoded. This reduces the complexity of the reduction function.

### D. Comparison against existing schemes

Next, we compared creation times for two fast MACs that can be used for Network Coding, results are summarized in Table IV. Execution times for these constructions are in the order of $\mu$s as well; the results show the execution times for the creation of exactly $1$ MAC and verification of that MAC for a generation of $16$ packets. Yu et al.'s construction from [6] changes its execution time, depending on the number of packets that have been coded, verification time for this scheme consider an uncoded packet, which is its fastest case. MAC generation times are slower than our IDNC proposal; however, they are significantly faster than our proposal for the small generation variant. Regarding security, in the case of Agrawal et al.'s [4] construction, many MACs must be computed since the output of each MAC is only one bit long, which reduces performance for both generation and verification in real scenarios, where several keys must be used. Yu et al.'s construction [6] can produce individual MACs with larger security, but still several of them are needed, since a single MAC does not protect all the positions in the payload. Both sytems also suffer from drawbacks associated to their MAC nature, such as key distribution and the overhead vs collussion resistance trade-off among others.

In terms of verification times, for all our variants they were in the order of $5$ $\mu$s for MD5 and $7$ $\mu$s for SHA. This results

Table IV. EXECUTION TIMES FOR CREATION AND VERIFICATION OF EXISTING SCHEMES WITH 1500 BYTE PACKETS

| Scheme | Generation ($\mu$s) | Verification ($\mu$s) |
|---|---|---|
| Agrawal et al. [4] | 164 | 184 |
| Yu et al. [6] | 216 | 209 |

apply to a network of any size with any number of attackers; this result does not include the time needed to authenticate the packet defining the accumulator.

Regarding schemes using number theoretic functions [2], [3], [7], they were not considered in our simulation, since their performance is significantly slower than the MAC-based ones that were analyzed.

## VI. CONCLUSIONS

We presented a proposal with 3 variants to prevent pollution in XOR Network Coding, unlike existing proposals in the literature, our construction does not rely on homomorphic functions and can be instantiated using standard primitives in a practical scenario. On the three variants, verification is in the order or $\mu$s due to the use of fast cryptographic accumulators. The limitations of the variants are related to the size of generation or the kind of coding strategy they can handle. The first variant is not concerned with the coding strategy used to forward packets, but can only be used for small generations.

The second one does not have a limit on the gerenation size, but can can only be used in networks where Immediately Decodable Network Coding is used; our tests outperforms existing constructions tailored at XOR Network Coding in both generation and verification times. The final variant allows a more diverse coding strategy, but for realistic scenarios is confined to small generations; to achieve this diversity, we selected packets according to a pre-defined strategy that bounds the number of packets regardless of the type of XOR Network Coding used by the network.

The proposal presents an interesting trade-off between the source generating the content and nodes forwarding it, this is useful for networks where relays and sinks have considerably less capabilities than the source. As future work we pretend to explore what additional cryptographic accumulators, hash functions and strategies for adding combinations to the accumulator can be used to extend the proposal.

## REFERENCES


[1] P. McDaniel, and S. McLaughlin. "Security and privacy challenges in the smart grid." *IEEE Security Privacy 7.3* pp. 75–77, 2009.

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

[3] D. Boneh, D. Freeman, J. Katz, and B. Waters, "Signing a linear subspace: Signature schemes for network coding," in *Public Key Cryptography–PKC 2009*. Springer, pp. 68–87, 2009.

[4] S. Agrawal and D. Boneh, "Homomorphic macs: Mac-based integrity for network coding," in *Applied Cryptography and Network Security*. Springer, pp. 292–305, 2009.

[5] P. Zhang, Y. Jiang, C. Lin, H. Yao, A. Wasef, and X. Shen, "Padding for orthogonality: efficient subspace authentication for network coding," in *INFOCOM, 2011 Proceedings IEEE*. IEEE, pp. 1026–1034, 2011.

[6] Z. Yu, Y. Wei, B. Ramkumar, and Y. Guan, "An efficient scheme for securing xor network coding against pollution attacks," in *INFOCOM 2009, IEEE*. IEEE, pp. 406–414, 2009.

[7] D. Boneh and D. M. Freeman, "Homomorphic signatures for polynomial functions," in *Advances in Cryptology–EUROCRYPT 2011*. Springer, pp. 149–168, 2011.

[8] J. Dong, R. Curtmola, and C. Nita-Rotaru, "Practical defenses against pollution attacks in wireless network coding," *ACM Transactions on Information and System Security (TISSEC)*, vol. 14, no. 1, p. 7, 2011.

[9] A. Le, A. S. Tehrani, A. G. Dimakis, and A. Markopoulou, "Instantly decodable network codes for real-time applications," *arXiv preprint arXiv:1303.7197*, 2013.

[10] L. Keller, E. Drinea, and C. Pragouli, "Online broadcasting with network coding," in *Network Coding, Theory and Applications, 2008. NetCod 2008. Fourth Workshop on*. IEEE, 2008, pp. 1–6.

[11] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Médard, and J. Crowcroft, "Xors in the air: practical wireless network coding," in *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 4. ACM, pp. 243–254, 2006.

[12] C. Bron and J. Kerbosch, "Algorithm 457: finding all cliques of an undirected graph," *Communications of the ACM*, vol. 16, no. 9, pp. 575–577, 1973.

[13] J. Benaloh and M. De Mare, "One-way accumulators: A decentralized alternative to digital signatures," in *Advances in CryptologyEUROCRYPT93*. Springer, pp. 274–285, 1994.

[14] N. Fazio and A. Nicolosi, "Cryptographic accumulators: Definitions, constructions and applications," *Paper written for course at New York University: www. cs. nyu. edu/nicolosi/papers/accumulators. pdf*, 2002.

[15] R. C. Merkle, "A certified digital signature," in *Advances in CryptologyCRYPTO89 Proceedings*. Springer, pp. 218–238, 1990.

[16] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[17] K. Nyberg, "Fast accumulated hashing," in *Fast Software Encryption*. Springer, pp. 83–87, 1996.

[18] C. Karlof, N. Sastry, Y. Li, A. Perrig, and J. Tygar, "Distillation codes and applications to dos resistant multicast authentication," in *Proceedings of the ISOC Symposium on Network and Distributed System Security (SNDSS)*. Citeseer, pp. 37–56, 2004.

[19] J. M. Park, E. K. Chong, and H. J. Siegel, "Efficient multicast packet authentication using signature amortization," in *Security and Privacy, 2002. Proceedings. 2002 IEEE Symposium on*. IEEE, pp. 227–240, 2002.

[20] R. Rivest, "The md5 message-digest algorithm," 1992.

[21] FIPS, "180-2 federal information processing standards publication," *SECURE HASH STANDARD, National Institute of Standards and Technology*, 2002.