SinPack: A Security Protocol for Preventing Pollution Attacks in Network-Coded Content Distribution Networks

Wassim Itani César Ghali Ahmad El Hajj Ayman Kayssi Ali Chehab
Department of Electrical and Computer Engineering
American University of Beirut
Beirut, Lebanon
{wgi01, csg04, ame34, ayman, chehab}@aub.edu.lb

Abstract- We present SinPack, a security protocol for preventing packet pollution attacks in network-coded content distribution networks. SinPack employs a homomorphically-addressable Bloom filter data structure to enforce the integrity of network-coded packets all the way from source to destination. Using a Bloom filter “amortizes” the functionality of traditional cryptographic integrity verification constructs (Message Authentication Codes, hash trees, digital signatures, etc) in a relatively small-sized data structure. This aids in reducing network traffic and, more significantly, allows the incremental integrity verification of out of order network packets. The novel homomorphic Bloom filter construction permits intermediate routers and destination end systems to verify the integrity of source packets even after being network-coded by routers. This methodology avoids the need to establish expensive and intricate trust relationships among the different network routers and ensures the authenticity of the integrity structures using a single source public-key operation. Moreover, SinPack not only allows the content downloader to immediately verify the integrity of coded packets, but also provides this capability to any intermediate router on the path to the destination. This helps in eliminating polluted packets in the network upstream closest to the source of attack and as a result contributes to a great reduction in bogus network traffic and hence sizeable energy savings.

Keywords-network coding security; Bloom filter; pollution attacks; homomorphic hash functions

I. INTRODUCTION

Network coding has brought several advantages over traditional store-and-forward communication networks, such as increasing throughput, minimizing energy consumption, and reducing latency. However, these kinds of networks are prone to several security and privacy threats which jeopardize the integrity of network operations, the confidentiality of encoded packets, and the anonymity of network flow.

The network coding theory was first presented in [1]. In this paper Ahlswede et al define a network coding node as a network node that is capable of processing data packets and performing various coding operations on their contents. This makes such emerging networking architecture revolutionary compared to the way packets are handled in current communication networks.

Today, network nodes or routers follow the store-and-forward model where data packets are stored in input/output queues, copied, and forwarded over the appropriate link interface(s). The only processing operation applied on the input packets is represented in the replication mechanism. Good references on this topic are presented in [2, 3, 4].

Similar to any networking architecture, network coding systems suffer from many potential security vulnerabilities and attacks. The most dangerous type of attack is referred to as the pollution attack where a malicious network router injects bogus packets into the network to disturb the network coding process. The bogus input packets to a network coding node will pollute the buffers resulting in the production of unpredictable output packets. If this kind of attack is not detected and accounted for in the vicinity of the source of attack, its disastrous effects will propagate to pollute the coded packets of the whole network.

In this paper we present SinPack, a pollution prevention protocol for network coding-capable networks. SinPack operates on network routers to ensure the integrity of network-coded packets all the way from source to destination. The integrity mechanisms employed rely on a probabilistic set membership data structure known as the Bloom filter [21]. Using a Bloom filter aids in reducing network traffic, verifying the integrity of out of order packets, and tolerating high packet loss rates. SinPack customizes the standard Bloom filter construction with novel homomorphic addressing building blocks to support the integrity enforcement of native, as well as, network coded packets. This methodology avoids the need to establish and maintain expensive trust relationships among the different network routers. The SinPack protocol can be executed on any intermediate router on the path from source to destination(s) so as to isolate the pollution packets as close as possible to the source of attack. The intermediate routers contribution in the pollution prevention mechanism leads to a great reduction in bogus network traffic and hence sizeable energy savings.

The rest of this paper is organized as follows: Section II presents a literature survey of the main protocols related to the proposed work. Section III provides a brief background overview of the cryptographic mechanisms and data structures used in the protocol design. The system model is presented in Section IV. Section V describes the simulation performance results obtained when testing SinPack on a simulated network topology. Conclusions are presented in Section VI.

II. RELATED WORK

A large amount of research work has dealt with the pollution attack problem in network coding. A major portion of this research work added different forms of cryptographic integrity enforcement constructs to each network packet to verify its integrity and to reject any maliciously crafted/injected packets.

Some of the cryptographic techniques relied on symmetric-key cryptography to enforce the integrity of packets [5] mainly by adding a Message Authentication Code (MAC) to each source-generated packet. These symmetric-key based solutions depend on the presence of a secure key distribution mechanism. Symmetric-key distribution has proved to be infeasible and not scalable in large networks [23].
Other cryptographic solutions adopted pure public-key mechanisms to authenticate source packets [6, 7]. The general framework here is to sign the source packets with the sender’s private key and have each recipient check the integrity and authenticity of the incoming packets by verifying the signature using the sender’s public key. These solutions are hindered by the extremely resource-intensive public-key operations which drastically waste the performance gains achieved by network coding.

To get the authentication and integrity advantages of public-key cryptography while reducing the computational and network overhead, signature amortization techniques [8, 9, 10] have been proposed in the literature. The main idea here is to be able to authenticate a block of packets using a single public-key signature. In [8], Gennaro and Rohatgi presented a technique for authenticating digital streams using hash chains. The hash chain is recursively constructed over a block of packets and the final value in the hash chain is digitally signed to ensure the integrity of the whole chain. This method cannot tolerate packet loss and requires the orderly delivery of packets.

Another popular signature amortization technique is represented in the use of Merkle hash trees [11, 12]. In this approach, the block of packets to be authenticated occupy the leaf nodes of the tree and are consecutively hashed to produce the sibling nodes up to the root hash node. The hash value at the root of the Merkle tree is digitally signed to authenticate the whole tree hash structure. In this approach, the source packets are appended with the signed root value and the tree hash values required to reconstruct the hash chain from the corresponding tree leaf up to the root (refer to [14] for more information about the Merkle hash tree construction). The main problem with this approach is the processing overhead required to verify the packet authenticity and the overhead information added to each source packet. This overhead is logarithmically dependent on the number of packets in the authenticated block.

Another class of signature amortization schemes is the one that relies on erasure codes [13, 14, 15]. The idea behind erasure codes is very simple: an encoder at the sender encodes the block of packets into a set of segments and adds redundant message blocks to the original segments. The redundancy information at the receiver allows it to recover the original blocks despite the loss of an upper bound number of packets.

In [14], Krohn, Freedman, and Mazieres presented an erasure coding scheme that allows the receiver to immediately verify the integrity of encoded blocks, thus saving bandwidth and preventing the pollution of the download cache. This technique sends a hash tree of the source file to intermediate nodes. The hashing relies on homomorphic hash functions to allow intermediate nodes to calculate the hash of network-coded packets from the hashes of the source packets. The main disadvantage of this scheme is its high computational cost and the sizeable bandwidth overhead due to the use of hash trees.

In [16], Gkantsidis and Rodriguez presented the GR scheme which, based on [14], also relies on homomorphic hash functions. However, the GR scheme requires a separate secure channel for delivering the source packet hashes to the intermediate routers. This requirement is based on a very strong security assumption which renders this research impractical. Charles, Jain, and Lauter presented a new homomorphic signature technique (the JCL scheme) that relaxes the secure channel GR requirement. However, this JCL scheme is based on complex and computationally expensive Weil pairing transformations [17] over elliptic curves. It should be mentioned here that several other approaches were proposed which rely on information theoretic principles for preventing packet pollution in the receiver’s buffers. Of these, we can mention the work by Ho in [18] and Jaggi in [19]. The main problem with such approaches is that they can only filter polluted packets at the receiving nodes and not at the intermediate routing nodes. This dramatically increases the energy consumption and bandwidth waste if the pollution attacker resides in the network upstream near the source node.

A recent research work presented in [20], employs the Null Space of network coded packets to filter out polluted packets without relying on homomorphic hash functions. The main disadvantage in this paper is due to the assumption that the attacker is unaware of the network topology and thus unable to identify the Null Keys. We believe that this assumption represents a very weak attacker model which renders this work impractical in real networks.

III. BACKGROUND

The integrity-enforcement mechanism employed in SinPack mainly relies on a probabilistic data structure known as the Bloom filter [21]. A Bloom filter is a space-efficient data structure that is used to perform set membership verification using low-cost hash functions in constant time. The main components of a Bloom filter are:

- An $m$-dimensional bit array initially set to zero
- A set of $k$ different hash functions that map an input element to $k$ different addresses in the $m$-bit array.

An element is added to the Bloom filter by generating a set of $k$ addresses using the $k$ hash functions. These addresses are mapped to $k$ positions in the $m$-bit array. The bit value of the mapped addresses is set to 1 in the array. To test for set membership, the element is input to each of the $k$ hash functions to get $k$ array addresses. The bit values of the $k$ addresses are fed into an AND logical function. If the result of the AND operation is 0 (any of the bits at these positions is 0), then the element is not in the set. Otherwise, if all bit values are 1, the element is in the set with a very high probability.

Set membership false positives may rarely occur if for instance the $k$ bits have been set to 1 during the addition of other elements to the Bloom filter. The false positives probability increases with the number of elements added ($N$) to the Bloom filter and decreases with the number of bits $m$ in the Bloom filter bit array, as follows [22]:

$$
\Phi \approx (0.6185)^{\frac{n}{m}}
$$

IV. SYSTEM DESIGN

This section presents an overview of SinPack system design and protocol operation. The design relies on a Bloom filter authentication structure whose hash addressing blocks are customized based on homomorphic encryption functions.

A. Design Goals

The main design goals of the pollution prevention mechanism are summarized in the following points:

1. Reducing network traffic imposed by the integrity solution by adopting a small-sized authentication data structure.
2. Allowing the incremental verification of out-of-order network packets.
3. Maintaining the proper operation of the pollution prevention protocol even in the case of high packet loss rates in the network.
4. Enabling intermediate network routers and destination end-systems to verify the integrity of native packets (packets generated by the source content-distribution node) as well as

978-1-4244-5637-6/10/$26.00 ©2010 IEEE
as network-coded packets (packets obtained by a linear network-coding transformation at routers).

5. Reducing network-wide energy and bandwidth consumption by dropping polluted traffic as close as possible to the pollution attack source. This is achieved as a side-effect of design goal 4.

B. Homomorphically-Addressable Bloom Filter Design

To obtain a homomorphically-addressable Bloom filter from a traditional Bloom filter, we replace the hash functions by homomorphic encryption functions. To generate a different Bloom filter blocks, the same homomorphic encryption function is used but with a different encryption key. Note that in our Bloom filter construction, we do not really need the Bloom filter blocks, the same homomorphic encryption function is used to produce a different combination. This is achieved by factorizing inputs from the address of their network-coded linear combinations. This is achieved by factorizing inputs from the address of their network-coded linear combinations.

By employing such a construction in the Bloom filter address space, the Paillier homomorphic encryption function [24] is used to generate the k Bloom filter address spaces. The Paillier homomorphic cryptosystem is described by the following equations:

\[ E(K_{PU}, pk_1) \times E(K_{PU}, pk_2) \times \cdots \times E(K_{PU}, pk_j) = E(K_{PU}, f(pk_1, pk_2, \ldots, pk_j)) \]  

Where \( E(K_{PU}, pk) = (g^{pk} \times p^n) \mod n^2 \)

\( K_{PU} = (g, n) \) and \( n = p \times q \), \( p \) and \( q \) are two large primes, \( g \in \mathbb{Z}_{n^2}^\star \), \( r \in \mathbb{Z}_n \), and \( pk \) is the input packet to the Bloom filter. \( f(pk_1, pk_2, \ldots, pk_j) \) is a weighted linear combination of the \( j \) input packets.

By employing such a construction in the Bloom filter address generation, we are able to calculate the addresses of individual inputs from the address of their network-coded linear combination. This is achieved by factorizing \( E(K_{PU}, f(pk_1, pk_2, \ldots, pk_j)) \) into its constituent elements \( E(K_{PU}, pk_j), E(K_{PU}, pk_2), \ldots, E(K_{PU}, pk_1) \). Since the factorization is not unique and may produce more than one result, we enforced a one-to-one bijection function using a factorization oracle. The oracle algorithm is presented below:

```
Algorithm 1: Factorization Oracle

Inputs: \( \mathcal{F}_{kx1}(K_{PU,x1}, f(pk_1, pk_2, \ldots, pk_j)) \)

Output: \( \mathcal{BF}_{mx1} \) if the network-coded packet belongs to the Bloom filter, otherwise false
```

It is worth mentioning that the factorization oracle described above exponentially reduces the false positive probability with increasing \( j \). The significance of this result is noticeable when checking the integrity of network-coded packets resulting from the transformation of a relatively large number of native packets, i.e., large \( j \). This will greatly enhance the security level of the integrity solution as will be discussed in Section V.

We summarize this result in the following self-contained lemma.

Lemma 1. The application of the factorization oracle on a network-coded packet, for verifying the existence of its native components in the Bloom filter, reduces the false positives probability exponentially with the number of native components in the network coded transformation.

Proof. Let \( \phi \) be the Bloom filter false positives probability when testing the set membership of a single native packet and \( j \) be the number of native components in the network coding transformation. Since the factorization algorithm verifies the set membership of \( j \) independent elements to produce a true result, the overall false positives probability will be the product of the false positives probabilities of the \( j \) independent elements, i.e., \( \phi^j \). This completes the proof of Lemma 1.

C. Protocol Architecture and Operation

The network model assumed in this work consists of a set of source content distribution nodes connected to a set of destination nodes via a network-coded network. The network is composed of a collection of routers capable of performing linear network coding operations on packets. The SinPack protocol operates to prevent any pollution attack on the contents of a file as it is being transferred from a source node to a set of destination nodes. In securing the transmission of the file from the source to the destinations, the SinPack protocol steps are described as follows:

1. The source node divides the file \( F \) into a set of equal-sized partitions each comprised of \( N \) packets (padding is employed if the file size is not a multiple of the size of a file partition).
2. For each file partition \( i \), a homomorphic Bloom filter data structure is created. The encryption keys \( K^{i}_{PU,k} = \{K^{i}_{PU,k}, K^{i}_{PU,k'}, \ldots, K^{i}_{PU,k}\} \) are randomly generated by the source node.
3. The set of packets \( pk_{k1} \rightarrow pk_{ni} \) comprising the file partition \( i \) are added to the Bloom filter.
4. The source node transmits the structure shown in Figure 2 over the network-coded network (the file consists of \( N \times R \) packets).
5. Whenever an intermediate router receives a packet
containing the Bloom filter for a certain file partition and its corresponding homomorphic keys, it verifies its authenticity and integrity using the supplied source digital signature. This makes the intermediate router prepared to check any packet belonging to this file partition against the corresponding Bloom filter. It is assumed here that the packet partition number and its type (native or network-coded) are encoded in certain fields in the packet header.

6. Every time an intermediate router receives a packet it firstly determines its type and partition number from the packet header. If the received packet is native, the intermediate router tests it against the partition’s Bloom filter. If the filter returns true, then the protocol accepts the packet with a very high confidence level. This confidence level is the complement of the false positives probability of the Bloom filter. As will be shown in Section V, the false positives probability is kept very low by tuning the parameters represented by the filter size and the number of packets per file partition. On the other hand, if the filter returns false, then the router drops the packet by considering it a polluted packet. If the received packet is network-coded consisting of a linear combination of native packets, say \( pk_1, pk_2, \ldots, pk_j \), the intermediate router calculates the addresses generated by the Bloom filter’s homomorphic encryption functions. These addresses are represented by the vector \( \mathbf{E}_{kx} = \left(\mathbf{K}_{pukx}, \cdot f(pk_1, pk_2, \ldots, pk_j)\right) \). This vector is fed to the Factorization Oracle algorithm described in Section IV B. If the algorithm produces a true result, the packet is accepted with a high confidence level. This level is equivalent to the complement of the false positives probability of the Bloom filter which exponentially decreases with \( j \). To enhance the performance, scalability, and interoperability of the pollution-prevention protocol, SinPack does not mandate the execution of the protocol at every intermediate router on the path to the destination. That is, routers may optionally forward the packets to their next hop addresses without executing the protocol steps. This is considered crucial in cases where the intermediate router is overloaded or the Bloom filter packet arrival is delayed. However, running the protocol is highly recommended to filter out polluted packets as close as possible to the source of attack.

7. At destination nodes, the same SinPack protocol steps described above must be executed to ensure the integrity of all the file partitions.

Note that the integrity verification mechanisms employed in the SinPack design do not depend, in any way, on the packets delivery order. This allows the verification of out of order packets. Moreover, having independent packet-level integrity verification makes the protocol immune against high packet loss rates.

A very important parameter to assess the security of the pollution prevention mechanism is represented in the false positives probability of the Bloom filter when checking for the integrity of native and network-coded packets. Based on the formula presented in Section III and the argument in Section IV B, the filter false positives probabilities when handling native and network-coded packets are respectively given by the following formulas:

\[
\Phi_{\text{native}} \approx \left(0.6185\right)^m_N
\]

\[
\Phi_{\text{network-coded}} \approx \left(0.6185\right)^{m/j}_N = \Phi_{\text{native}}^j
\]

D. SinPack Design: From Theory to Practice

For the SinPack design to be applicable in modern network architectures, the protocol should be able to represent the packet bit patterns in relatively small integer values. This is very crucial for maintaining the efficiency of the Factorization Oracle algorithm in supporting high speed network traffic. To address this issue, an optimization step is added to the protocol design. On the source router, the SinPack protocol logically divides the contents of each native packet into \( s \) parts and feeds these parts into the Bloom filter instead of feeding the whole packet. On intermediate routers and destination nodes the verification process will be modified accordingly as follows: if the received packet is a native packet, then the router will test each of its \( s \) parts against the Bloom filter, only if all \( s \) parts belong to the initial source set then the packet is accepted. On the other hand, if the packet is network-coded, then each of the \( s \) packet parts is fed into the Factorization Oracle. The packet is considered valid if all the \( s \) factorization executions return true.

It should be noted here that logically dividing the packet not only allows the factorization algorithm to handle small integer values, but also lays the ground for further optimizations that are presented in Section V.

A very interesting property here is represented in the fact that the logical packet division mechanism does not affect the false positives probability of the overall system despite the fact that the initial Bloom filter set has increased by a factor of \( s \). This is presented in the following lemma.

Lemma 2. The logical packet division process does not affect the false positives probability of the overall system and hence the security level of the pollution prevention protocol.

Proof The logical packet division increases the source Bloom filter set from \( N \) to \( s \times N \) and thus the \( m/N \) term in Equations 4 and 5 is divided by \( s \). Moreover, since the integrity verification unit is now a packet part instead of a whole packet, the overall false positives probability in the native case as well as in the network-coded case will be the multiplication of the false positives probabilities of the \( s \) packet parts. This causes the \( m/N \) term in Equations 4 and 5 to be multiplied by \( s \). As a result, the \( s \) term in the numerator cancels that in the denominator and thus the native and the network coded probabilities are not affected. This completes the proof of lemma 2.

E. Pollution Prevention: The Sooner The Better

The SinPack pollution prevention mechanism gives intermediate routers the ability to filter out polluted packets so as to isolate the effects of the attack as close as possible to the source of the attack. This results in the reduction of bogus network traffic and hence leads to a major reduction in network-wide energy consumption. To illustrate this fact, we consider the reference
Butterfly network (Figure 3). Assume that source $S$ wants to distribute an $X$ byte file at a rate of $\alpha Bps$ to destinations $D_1$ and $D_2$, and that a pollution attack source exists on the $S - R$ link injecting $\beta Bps$ of bogus packets. The effect of the initial location of the pollution prevention mechanism on the increase of bogus traffic is illustrated in Table 1. For simplifying the analysis, we assume high capacity links that can accommodate the flows from the attacker and the source.

TABLE 1. Effect of Pollution Detection Location on the Increase in Bogus Network Traffic in Bytes

<table>
<thead>
<tr>
<th>Link</th>
<th>Detection Router</th>
<th>$N_1$</th>
<th>$N_2$</th>
<th>$D_1$ or $D_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S-R$</td>
<td>$\frac{\beta \times X}{\alpha}$</td>
<td>$\frac{\beta \times X}{\alpha}$</td>
<td>$\frac{\beta \times X}{\alpha}$</td>
<td>$\frac{\beta \times X}{\alpha}$</td>
</tr>
<tr>
<td>$R-N_1$ &amp; $R-D_2$</td>
<td>$-\frac{2 \times \beta \times X}{\alpha}$</td>
<td>$\frac{2 \times \beta \times X}{\alpha}$</td>
<td>$\frac{2 \times \beta \times X}{\alpha}$</td>
<td></td>
</tr>
<tr>
<td>$N_1-N_2$</td>
<td>$-\frac{\beta \times X}{\alpha}$</td>
<td>$\frac{\beta \times X}{\alpha}$</td>
<td>$\frac{\beta \times X}{\alpha}$</td>
<td></td>
</tr>
<tr>
<td>$N_2-D_1$ &amp; $N_2-D_2$</td>
<td>$-\frac{\beta \times X}{\alpha}$</td>
<td>$-\frac{\beta \times X}{\alpha}$</td>
<td>$\frac{2 \times \beta \times X}{\alpha}$</td>
<td></td>
</tr>
<tr>
<td>Overall Overhead in Bytes</td>
<td>$\frac{\beta \times X}{\alpha}$</td>
<td>$\frac{3 \times \beta \times X}{\alpha}$</td>
<td>$\frac{4 \times \beta \times X}{\alpha}$</td>
<td>$\frac{6 \times \beta \times X}{\alpha}$</td>
</tr>
</tbody>
</table>

Table 1 shows that the pollution network traffic decreases when the filtration mechanism is executed close to the source of the attack. For example, if on the path from the source to the destinations, router $N_1$ initiates the detection process the overhead traffic on links $S-R$, $R-N_1$ and $R-D_2$ is $(\beta \times X)/\alpha$ bytes respectively, thus resulting in an overall traffic of $3 \times (\beta \times X)/\alpha$ bytes. The traffic savings will be much more appreciated in relatively large networks with a very large number of intermediate routers.

F. SinPack Design Parallelism

The SinPack protocol design natively exposes a significant degree of parallelism which makes it a suitable candidate for parallel hardware implementation. This design property will dramatically enhance the protocol performance by employing parallel modes of operation. The main parallelizable operations in the SinPack design are summarized in the following points:

1. The Bloom filter set update primitive can be achieved in $k$ parallel homomorphic encryption and memory write operations.
2. The Bloom filter set membership verification primitive can be achieved in $k$ parallel homomorphic encryption and memory fetch operations.
3. The Factorization Oracle algorithm can be easily parallelized to achieve better real-time computational performance.
4. The integrity verification of individual packet parts can be executed in isolation of the other parts. This requires the output of $s$ parallel and independent verification operations to be combined to validate the integrity of the whole packet. Future work will show a parallel hardware implementation of the system using a Field-Programmable Gate Array (FPGA).

V. PERFORMANCE ANALYSIS AND RESULTS

This section presents the simulation model we considered to experimentally test the pollution prevention protocol performance. A sample Butterfly network, extended with a set of network coding routers, is simulated using the $C#/.Net$ language. The physical machine used is an Intel(R) Core(TM) i7 x64 CPU running at 1.60 GHz and supported with 4 GB of RAM. The operating system used is Windows 7 Home Premium. Every node in the network is modeled using a $C#$ class and the communication among the different network nodes is achieved using .Net interclass communication. The network nodes forwarding tables are preconfigured with static routes. The sample SinPack protocol implementation was tested with the parameters presented in Table 2.

TABLE 2. SinPack Protocol Simulation Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of network nodes</td>
<td>20</td>
</tr>
<tr>
<td>$BF$ size (m) in bits</td>
<td>11376</td>
</tr>
<tr>
<td>Number of Homomorphic encryption blocks ($k$)</td>
<td>3</td>
</tr>
<tr>
<td>Number of packets per network-coding transformation ($j$)</td>
<td>2</td>
</tr>
<tr>
<td>Number of packets per file partition ($N$)</td>
<td>700</td>
</tr>
<tr>
<td>Size of the packet in bytes</td>
<td>400</td>
</tr>
<tr>
<td>Logical packet part size in bytes</td>
<td>2</td>
</tr>
</tbody>
</table>

A key design parameter is the logical packet part size. We found out that an 16-bit part would provide the optimum performance and compatibility due to three main reasons: Firstly, an 16-bit part size makes the protocol compatible for operation on an assortment of processing architectures supporting a wide range of computing devices. Secondly, this size allows for the efficient implementation of the factorization algorithm by reducing the size of the integers handled by this algorithm. Finally, a 16-bit size facilitates the pre-computation of the factorization process results and allows accessing them using very fast lookup and indexing operations.

Based on a set of 15 simulation runs using a lookup table implementation of the factorization algorithm, the average time needed by the simulated software router class to verify the integrity of a native packet is 1.41 ms, while the average time to verify the integrity of a network-coded packet is 2.23 ms. Actual performance figures from a hardware implementation on an FPGA with all the parallel optimizations will be presented in future work.

The exponential decay of $\Phi_{\text{network\_coded}}$ with $j$, significantly enhances the security level of the integrity solution and greatly limits the possibility of any successful pollution attack resulting from injecting maliciously crafted network packets. This fact is emphasized in the plots presented in Figure 4. Figure 4a illustrates the variation of $\Phi_{\text{native}}$ and $\Phi_{\text{network\_coded}}$ with the number of packets per file partition. For a given Bloom filter size, $N$ and $j$ are the two main parameters affecting the false positives probability. As shown in the figure, the false positives probability increases with $N$ and decreases with $j$. Hence $N$ and $j$ can be fine-tuned to achieve the desired system false positives probability. For instance, to achieve a false positives probability of $\approx 10^{-15}$ with a network code consisting of two packets ($j = 2$), the value of $N$ should be set to $\approx 320$.
packets. With a network code consisting of three packets \((j = 3)\), \(N\) should be set to \(\approx 500\) packets. The value of \(j\) depends primarily on the network coding transformation implemented and thus \(j\) is not directly controlled by the SinPack protocol on the routing nodes. \(N\), on the other hand, can be directly set and adjusted by the SinPack protocol configuration to realize high integrity confidence levels. However, it should be noted here that decreasing \(N\) to very small values will incur noticeable performance degradation at the routing nodes. This is due to the fact that decreasing \(N\) results in increasing the number of file partitions \(R\), and consequently increases the number of public-key digital signatures required to authenticate the Bloom filters associated with the \(R\) file partitions (see Section IV C).

In addition to the small false positives probabilities obtained, the possibility of a successful pollution attack is reduced considering the structure of modern network packets. For example, in IP networks, it is highly infeasible to craft a packet with valid header fields and checksum that successfully passes the Bloom filter test.

Figure 4b presents the effect of increasing the number of packets per network-coding transformation \((j)\) on the Bloom filter false positives probability. We can note that \(\Phi_{\text{network coded}}\) asymptotically approaches 0 with relatively small values of \(j\) \((j = 5, \Phi_{\text{network coded}} \approx 10^{-17})\). This result is significant in networks with sophisticated network-coding transformations. This is due to the fact that the increase in the transformation entities aids in elevating the security level of the pollution prevention protocol.

VI. CONCLUSION

In this paper, we presented SinPack, a novel pollution prevention protocol in network-coded networks. SinPack leverages a homomorphically-addressable Bloom filter construction to secure the integrity of network packets end-to-end from source to destination. SinPack is capable of verifying the integrity of native source packets as well as network-coded packets, handling out-of-order packets, and tolerating high packet loss rates. Simulation results show the feasibility of applying SinPack to prevent pollution attacks in network-coded networks. Future work includes the implementation of SinPack in hardware on FPGA-based network nodes.

ACKNOWLEDGMENT

The authors would like to thank and acknowledge the support of the Lebanese National Council for Scientific Research and the AUB University Research Board.

REFERENCES