On the Integrity of Network Coding-based Anonymous P2P File Sharing Networks

Paolo Gasti
DISI – Università di Genova
Via Dodecaneso, 35
16146 Genova (Italy)

Alessio Merlo
IEIIT–CNR
Via De Marini, 6
16149 Genova (Italy)

Giuseppe Ciaccio
DISI – Università di Genova
Via Dodecaneso, 35
16146 Genova (Italy)

Giovanni Chiola
DISI – Università di Genova
Via Dodecaneso, 35
16146 Genova (Italy)

***Abstract*—Network coding is a class of routing algorithms offering increased throughput and improved robustness to random failures. With traditional routing, intermediate nodes in the network may only forward unmodified packets. With network coding, instead, intermediate nodes are allowed to forward linear combinations of received packets. Original data can be reconstructed after collecting sufficiently many linear combinations. Current file sharing systems offer either low overhead and high bandwidth with no privacy, or acceptable privacy at very low speed. Thanks to network coding, a general-purpose P2P network can obtain a privacy/performance tradeoff that may be considered reasonable in most real-world scenarios. In this paper we present an integrity strategy for network coding-based P2P anonymous systems, specifically designed to preserve the anonymity of peers. Our approach is significantly easier to implement than current solutions when anonymity is required. We implement the cryptographic algorithms on which our method is based and provide performance figures. We also define verification strategies which use batching for improved performances together with an efficiency analysis.**

I. Introduction and Motivation

Peer-to-peer (P2P) networks have been designed to support data exchange among interconnected peer entities. In a P2P network, any peer acts both as a data requestor and a data provider. From a network perspective a peer accomplishes routing functionalities by sending data to neighbors.

P2P networks have been extensively studied as substrates for anonymous content distribution. One of the main goals is to make it unfeasible to identify the sender or the requestor of any block of data passing through the system. However, in these systems anonymity is usually obtained at the expense of poor performance: anonymous P2P networks are notoriously much slower than non-anonymous ones [21]. In general, the privacy/performance tradeoff of anonymous P2P networks is still an open research issue. It is worth noting that both anonymity and P2P networks always rely on sufficiently many peers cooperating in the network, but poor performance discourages users from joining; thus, good privacy/performance tradeoffs are of primary importance for these networks to be deployed and gain user acceptance.

Recent studies (e.g. [12], [23], [5], [6]) have shown that network coding as a routing technique is able to provide advantages in terms of throughput, delay, and fault tolerance in decentralized networks. The central idea of network coding is that any node in the network can forward random linear combinations of incoming packets instead of merely forwarding unmodified packets. This implies that individual blocks of data shall be routed onto many network paths, although in combination with data from other sources. One notable consequence is that redundant paths in the network are always exploited simultaneously rather than as alternatives. This leads to a potential increase of performance and fault tolerance, since there are more ways for a receiver to obtain a required data block. Moreover, different receivers may be actually sharing a common routing path where their requested data are travelling as linear combination bundles.

We believe that network coding could also be exploited to provide a degree of anonymity and deniability to user. Since data blocks of a file almost always travel linearly combined with other blocks from possibly different files, any peer may enjoy a degree of deniability for any given file whose data blocks are found to travel from/to it. Individual blocks may possibly serve to "cancel" unwanted content from other linear combinations, so downloading a block does not necessarily imply the will to enjoy the content associated with it. Network coding could in principle provide those guarantees while at the same time enhancing file sharing performance, thus encouraging people to join the system, with a consequent improvement in system performance and guarantees.

A major concern in systems that use network coding is posed by byzantine nodes that introduce corrupted data in the network. This behavior, known as "pollution attack" [5], allows an adversary to attack the system in a very effective way, because errors introduced into a single block can propagate and corrupt other data. This attack affects performance of a generic network, because individual nodes will waste bandwidth and time in downloading useless data. Thus, it is important to augment network coding with efficient solutions for checking the integrity of data blocks. This issue has been investigated in decentralized networks where a kind of block authentication can be assumed [16]. Basically, blocks are signed with the private key of the producer and this allows an easy form of integrity check. However this approach does not easily apply to anonymous networks.

In this paper, we tackle the problem of integrity in P2P networks based on network coding and providing anonymity to users. We believe that distributed and collaborative integrity checks could apply, provided they are specifically designed to suit anonymity requirements.

Anonymity of data producers prevents anybody to associate a block with the user who created it, so integrity checks based on block signatures are not applicable. Indeed, as keys are not to be associated with users, it becomes straightforward for a malicious peer to alter a block and sign it with its own signing key. Depending on the data encoding scheme, the corruption might become apparent only when the file is completely reconstructed, but by then several other possibly independent blocks would have already been polluted [13]. Even if a corrupted block is found by other peers, the malicious peer can deny the possession of the signing key and claim that the block was corrupted somewhere else. It is possible that this problem could be solved by associating a reputation with each public key; only keys with a reputation higher than a chosen threshold would be accepted as valid signing keys. This solution would have, however, several drawbacks. Signature produced by a new user who is sharing one or more files with other peers cannot be verified reliably until a certain threshold for its reputation has been met. Another problem of this approach is that reputation values for a specific public key should be made available to all peers. Failing to do so would imply that a key could have a high reputation for some peers, and a low one for others. But a global knowledge would require a P2P information exchange protocol (see e.g. [1]), which is in turn susceptible to integrity attacks; alternatively, a centralized database with the association of a key and its reputation should be maintained, which may be impractical in most circumstances.

Another way to protect packet integrity in P2P systems based on network coding is to use homomorphic hash algorithms. However, the integrity schemes based on homomorphic hashes proposed so far do not take into account anonymity. They indeed require a trusted channel between the peers and a database which stores the hash values of each packets. In the case of anonymous P2P, this channel must be anonymous as well. Such a secure *and* anonymous channel, in turn, poses new availability and integrity problems. Availability could be enhanced by replicating the hash database, but this would increase the hash integrity problem in case some of the replicas are managed by malicious users.

II. Motivation and contribution

We propose a new solution for integrity checking on anonymous P2P networks based on network coding. More specifically, the main contributions of this paper are: (1) definition of an identification scheme for data blocks; this identification scheme allows to overcome one of the biggest drawbacks of the solution proposed in [13], which is the need for a secure storage for the hash values of the blocks in the network and a secure authenticated channel for retrieving such hash values; and (2) an efficient strategy for checking the integrity of packets, both for a decentralized anonymous storage network based on network coding. We support our claims with performance evaluation of a small prototype which implements the integrity strategies described in this paper. Our idea is to abandon signature-based techniques altogether and use a simpler hash-based approach on the anonymous P2P network. Each new file is divided into blocks of a predefined size. A hash is calculated using a collision-resistant homomorphic hash function on each block individually. Thanks to the collision-resistance property of the hash function, different blocks have different hashes with overwhelming probability. We exploit this property to identify each block with its hash value. Peers willing to download a file retrieve the list of the IDs which compose such file. For each block they receive from their neighbors, the integrity is verified on the fly by calculating the hash value of the block and comparing it to the expected hash value. Corrupted blocks are identified immediately and do not pollute any other block. Instead of requiring a producer-specific information (the producer's public key), our proposal uses a global parameter which allows all peers to verify blocks.

We then extend the hash-based scheme to deal with blocks in batches, rather than one at a time, based on the assumption that faulty blocks are expected to be infrequent. This improves performance, but also requires efficient techniques for discovering corrupted blocks, and identifying byzantine neighbors, when presented with a faulty batch. Our scheme relies on the existence of a honest[1] super-peer that associates each shared file with the list of its blocks. We believe that this assumption is realistic, and certainly more realistic than assuming that each peer can efficiently and reliably associate a trust level to the public key of other peers. In our setup, peers only need to trust one entity – the super-peer – instead of several independent entities – the signers. Later in the paper we further discuss about this assumption and show that it is indeed realistic.

Similarly to other P2P file sharing systems [18], this approach allows a malicious content provider to share *fake* or corrupted files. Users will notice such attack only when a large enough portion of the file has been downloaded. However this well known problem is out of the scope of this paper, which focuses primarily on the integrity of the blocks within the overlay network.

The paper is organized as follows: Section III illustrates the state-of-the-art on integrity checking in different P2P networks. Section IV details the model of the anonymous P2P network we used for our proposal, while Section VI investigates our homomorphic hash algorithm, its extension to batches of data blocks, and its performance. In Section VII we discuss some strategies for isolating a corrupted block in a batch. Section VIII concludes the paper.

III. Related Work

Current anonymous or censorship-resistant P2P networks do not perform user authentication, do not trust third parties or central authorities, and try to unlink actions from users who perform them. Typically, such systems gain anonymity

[1]The super-peer does not try to subvert the system by maliciously alter the information that it is in charge of sharing. However it does not know any private information about the peers, such as private keys or their identities.

by repeatedly routing a message from one node to another, so as to gradually lose the sender-related information carried by a message along its way. This approach poses at least two challenges: queries and responses must be made unintelligible to intermediate nodes, the loyalty of which cannot be relied upon; at the same time, intermediate nodes should be made capable of filtering out corrupted messages, in order to confine a possible attack on data integrity. Thus, message integrity check and corruption confinement are two primary needs for these networks.

Some of the early anonymous P2P networks [8], [25], [22] did not address the problem very deeply: they relied on bare data replication to make integrity attacks more difficult (but not impossible) and ultimately turning them into attacks to performance. Other networks, however, implemented innovative data encodings that allow integrity checks and corruption confinement while coping with the distribution and anonymity requirements. For lack of space we only account for what is probably the most advanced one, namely, the encoding of GNUnet. GNUnet [4] is a distributed storage that uses an encoding called ECRS (Encoding for Censorship-Resistant Sharing [14]), an improvement over the original encoding of another famous anonymous distributed storage, namely, Freenet [7]. With GNUnet, files are segmented into fixed-size independent blocks, a feature that may greatly improve download performance by allowing so-called *file swarming*, that is, download of a file from more sources at a time. Each block B of data is individually encrypted under the symmetric key $H(B)$, then the encrypted block E(B) is stored in the distributed storage at the statistically unique address $H(E(B))$ (H is a cryptographic hash function). The information for retrieving B is thus the pair $\langle H(E(B)), H(B)\rangle$, called a Content Hash Key (CHK). The collection of CHKs denoting all file segments is then arranged in a logical tree, using indirection blocks similar to filesystem inodes which in the end converge to a root CHK. With GNUnet, a requestor node querying the CHK $\langle Q, K\rangle$ will send Q as a query; responses are of the kind $\langle Q, E(B)\rangle$, that is, the ciphered block travels back along with the query itself. Intermediate node cannot decipher the response, since they are oblivious of $K = H(B)$; yet, they can perform the integrity check (hashing $E(B)$ must yield Q), so tampered data can be filtered out very early, and an attack on data integrity essentially amounts to not forwarding data at all. The GNUnet encoding could not be easily extended to work with network coding; however it has indeed inspired us in devising our own integrity scheme.

Other anonymous networks, like Tor [9] or MorphMix [20], provide a multi-hop encrypted socket proxy service with no abstraction of a storage. Censorship resistance is not a primary goal with these systems, so pollution attacks are considered of lesser importance compared to attacks to anonymity. Message integrity in Tor is performed only at the two ends of each proxy circuit, by labelling each block of data with (the first four bytes of) the hash of the end-to-end session key (unknown to all other nodes along the circuit) combined with the data itself; intermediate nodes can thus pollute data and this will only be detected at the receiver end of the circuit, with some waste of resources along the circuit. MorphMix seems not to address integrity issues at all.

General purpose integrity schemes fall into two broad families, namely, those ones based on shared secret keys, and those ones based on public/private key pairs. Message Authentication Code (MAC) algorithms [15] fall into the former family, whereas Digital Signature (DS) algorithms [19], [10], [3] fall into the latter. None of them, however, is compatible with network coding: since packets are modified by intermediate nodes, neither MAC nor signature provide an appropriate solution. For this reason, a number of integrity mechanisms specifically designed for network coding have been proposed so far. In [5], Boneh et al. propose two signature schemes specifically designed to provide cryptographic protection against pollution attacks under realistic assumptions, i.e. even when the adversary can corrupt an arbitrary number of nodes in the network, eavesdrop on all network traffic, and insert or modify an arbitrary number of packets. The schemes in the paper ensure that the destination can filter out any corrupted packets and hence recover the correct file. The signature scheme is publicly verifiable, therefore intermediate nodes can discard corrupted packets as well (though whether this is actually done will depend on the computational resources of the intermediate nodes). The first scheme in the paper has constant public key size and per-packet overhead. The two schemes can be viewed as signing linear subspaces in the sense that a signature a subspace authenticates exactly the vectors in that subspace. In addition to the signature schemes, they also prove a lower bound on the signature length for any scheme for signing linear subspaces, showing that our constructions are essentially optimal in this regard.

Krohn et al. present in [17] an integrity verification scheme specifically designed for P2P networks that use rateless erasure codes for efficient multicast transfers. Their scheme is based on homomorphic hashing and closely mimics traditional P2P integrity checks, where each downloaded block is verified individually using cryptographic hashes or digital signatures. The scheme introduced in that paper is independent of encoding rate, therefore it is compatible with rateless erasure codes.

Gkantsidis and Rodriguez [13] propose a cooperative security scheme, based on the homomorphic hash function of Krohn et al. [17], where well-behaved users cooperate to protect themselves against malicious nodes by alerting affected neighbors when a corrupted block is found. The result of this is that even if each node checks for bad blocks infrequently, bad blocks do not propagate in the network with high probability. In this way the computation overhead for each node is greatly reduced at the cost of relying on a complex algorithm for cooperative verification. The applicability of their solution, however, is severely limited by the need of a secure channel, wich is used by peers to retrieve the hash values.

Yu et al. [26] extend the work of Gkantsidis and Rodriguez by proposing a signature-based scheme that allows the source to delegate its signing authority to forwarders. Forwarders can generate new signatures without interacting with the original source only if the newly encoded blocks are not corrupted. This result is achieved using a homomorphic signature scheme. Unfortunately Wang showed in [24] a vulnerability in the scheme of Yu et al. More specifically, he showed that the homomorphic property of the signature, together with the batch verification used in their scheme, allows an adversary to forge a signatures. Moreover, Gennaro et al. [11] note that Yu et al. incorrectly assume that $((A^b \bmod p)^d \bmod r) = (A \bmod p)^{bd} \bmod r$ for integers A, b, d, a, a prime p and an independent RSA composite r, therefore the scheme works only with negligible probability.

In the same paper [11], Gennaro et al. also propose their own homomorphic signature scheme for network coding. Their scheme is based on the RSA assumption, and they claim it is the first scheme which avoids bilinear groups and pairings for efficiency reasons. The bandwidth overhead is low for networks of moderate size and the scheme uses a public key of constant size. The paper also presents a homomorphic hash scheme that works modulo a composite. This scheme considers each information vector transmitted over the network as a single large integer. The collision resistance of the hash function is based on the factoring problem.

Our solution is based on the scheme presented in [17]. It is possible that the homomorphic hash defined by Gennaro et al. is also suitable for our purposes, although we haven't investigated on this yet.

IV. System Model

The main features of our system model are as follows:

- **Storage:** each peer can store received packets in a temporary buffer until it is ready to verify their integrity.
- **Communications:** peers can always reach one another via the underlying network, which is supposed not to introduce errors in the transmitted data. Network coding is used for encoding and routing individual packets in the network, as well as decoding packets once they have passed the integrity checks.
- **Secure channels:** each peer can run basic cryptographic primitives so that a possible eavesdropper cannot decode ciphertext or inject fake packets or alter in-flight packets in a communication channel between two peers. Therefore the integrity of each packet is completely under the responsibility of the sender.

The envisaged P2P system should provide some degree of anonymity to both the producer and the consumer of information. Due to this requirement, the integrity verification scheme should minimize any leakage of information about the producer of a verified block. Ideally, it should be even made impossible to determine whether two data blocks belonging to different files were produced by the same user. As already pointed out, this rules out signatures in general, and homomorphic signatures in particular (like, for instance, the scheme proposed in [5]). Indeed, although a key in itself may not leak information about its owners, signatures allows to determine whether two files have been signed by the same user (or, at least, by users who share a private key). Short-lived keys may not present such a drawback, but are at the same time unable to convey reputation information to the receiver, so they become virtually useless. Instead, it is much easier for a super-peer to build its reputation compared to a user's signing keypair, since we expect a super-peer to be a long-lived entity, well known by all peers, what is clearly not the case for most user's public keys.

V. Adversarial Model

The envisaged adversary has the following features:

- **Goal:** the goal of the adversary is to corrupt the content of some or all of the packets that flow through the network. Packets must be corrupted is such a way that legitimate peers cannot detect packet corruption until a significant portion of the file has been retrieved.
- **Compromise power:** the adversary can compromise any note at its will and fully control its behaviour. When compromised, a node will reveal all its secret information. The adversary can modify only the packets that flow through compromised nodes.
- **Parameter generation:** the adversary cannot interfere with the generation of the the global parameters for the system.
- **Defense awareness:** the adversary is fully aware of any scheme or algorithm that the peers use to defend the integrity of the packets they exchange.

VI. Homomorphic Hash for Network Coding

Homomorphic hash algorithms provide a valuable tool for verifying packet integrity when network coding is used. When a block is encoded, peers must attach the list of the IDs of the blocks used to produce that encoding. To verify an encoded block, a peer combines the list of IDs to obtain a single packet ID and then verifies the output of the homomorphic hash function on the aggregate block with the combined ID. Blocks with the same content belonging to different files will have the same hash and therefore appear as the same block to the system. Depending on the concrete instantiation of the network, this helps to increase either efficiency, because the same block does not have to be stored more than once even if it belongs to different files, or redundancy, since different blocks belonging to different files are implicitly replicated. We indicate the hash value for block b as ID_b. A homomorphic hash scheme suitable for our approach is defined by the following algorithms:

$\mathsf{Setup}(1^n)$. A probabilistic algorithm which, on input a security parameters 1^n, outputs the public parameter pk. The running time of the encryption, decryption, and the adversary algorithms are all measured as a function of a security parameter, which is expressed in unary notation.

$\mathsf{Hash}(pk, \mathsf{b})$. A deterministic algorithm which on input a public parameter pk and a block b, outputs a hash value ID. Given pk and b it must be computationally infeasible to find a block $\mathsf{b}' \neq \mathsf{b}$ such that $\mathsf{Hash}(pk, \mathsf{b}) = \mathsf{Hash}(pk, \mathsf{b}')$.

$\mathsf{Combine}(pk, \mathsf{b}_1, \ldots, \mathsf{b}_s, \mathsf{ID}_1, \ldots, \mathsf{ID}_{s'})$, $s \leq s'$. An algorithm which, on input a public parameter pk, a list of blocks $\mathsf{b}_1, \ldots, \mathsf{b}_s$ and a list of block IDs $\mathsf{ID}_1, \ldots, \mathsf{ID}_s$, outputs a new block $\mathsf{b}^{(s)}$ which is the linear combination of $\mathsf{ID}_1, \ldots, \mathsf{ID}_s$, and its hash $\mathsf{ID}^{(s)}$ derived from $\mathsf{ID}_1, \ldots, \mathsf{ID}_s$.

$\mathsf{Verify}(pk, \mathsf{b}, \mathsf{ID})$. An algorithm which, on input a public parameter pk, a block b and a hash ID outputs 1 if $\mathsf{Hash}(pk, \mathsf{b}) = \mathsf{ID}$, 0 otherwise.

While blocks have a fixed size, files are allowed to be of arbitrary size. Files are seen as an ordered sequence of blocks.

We use the global homomorphic hash algoritm proposed by Krohn et al. [17] for calculating the blocks ID and for integrity checks, since it produces relatively short hash output and allows batch verification for improved performances. The size of the hash values in this hashing scheme is independent of the encoding rate, although in our scenario an encoded block carries the list of all the identifiers of the blocks it was composed with. Moreover, hash values can be verified by the receiver on the fly. By allowing all the nodes to share the global hash parameters (global hashing), there is a single way to map a block b to $\mathsf{Hash}(\mathsf{b})$. Since we are using global hashing, copies of the same content independently published by different users look identical to the system.

A. The Hash Algorithm

In this section we briefly introduce the hash scheme of Krohn et al. [17]. Informally, two random primes p and q and a vector $\mathbf{g}$ are chosen by a trusted party and shared among peers. The trusted party can be the developer of the network coding library or one of the super-peers. The trust on this party is, however, limited since a malicious trusted party can generate collisions, which we assume can quickly be noticed. A collision is also the proof that such a malicious party exists and would lead to the re-generation of the public parameters.

Each file is divided into blocks, and each block is divided into several sub-blocks of size $|q| - 1$. The ID of a block has the same size of a sub-block. To combine two blocks, a peer simply computes the sum of the corresponding sub-blocks; let $\mathsf{b}_1 = (b_{1,1}, \ldots, b_{1,\ell})$ and $\mathsf{b}_2 = (b_{2,1}, \ldots, b_{2,\ell})$. The combination of b_1 and b_2 is $\mathsf{b}^{(s)} = (b_{1,1}+b_{2,1}, \ldots, b_{1,\ell}+b_{2,\ell})$. The combination of the identifiers ID_1 of b_1 and ID_2 if b_2 is $\mathsf{ID}^{(s)} = \mathsf{ID}_1 \cdot \mathsf{ID}_2$. An encoded block is not corrupted if $\mathsf{ID}^{(s)}$ is equal to the hash calculated on the combined block $\mathsf{b}^{(s)}$.

More formally, the three algorithms composing the homomorphic hash function are defined as:

$\mathsf{Setup}(1^n, 1^m)$. Picks two random primes p and q such that $|p| = n$, $|q| = m$ and q divides $(p-1)$. Then pick a vector $\mathbf{g} = (g_1, \ldots, g_\ell)$ composed of $\ell = \left\lceil \frac{|\mathsf{b}_j|}{m-1} \right\rceil$ random elements from $\mathbb{Z}_p$, all of order q. Output $pk = (p, q, \mathbf{g})$

$\mathsf{Hash}(pk, \mathsf{b})$. Divide each block b into ℓ sub-blocks $\mathsf{b}_1, \ldots, \mathsf{b}_\ell$ of size $|q| - 1$, then output a hash value ID calculated as:

$$\mathsf{Hash}(pk, \mathsf{b}) = \prod_{i=1}^{\ell} g_i^{\mathsf{b}_i} \pmod p$$

$\mathsf{Combine}(pk, \mathsf{b}_1, \ldots, \mathsf{b}_s, \mathsf{ID}_1, \ldots, \mathsf{ID}_s)$. Divides each block b_i into ℓ sub-blocks $b_{i,1} \ldots, b_{i,\ell}$ and calculate the combined block $\mathsf{b}^{(s)}$ as follows:

$j = 1 \ldots \ell$:

$$\mathsf{b}_j^{(s)} = \sum_{i=1}^{s} \mathsf{b}_{i,j} \pmod q$$

and its hash $\mathsf{ID}^{(s)}$ as

$$\mathsf{ID}^{(s)} = \prod_{i=1}^{\ell} \mathsf{ID}_i \pmod p$$

$\mathsf{Verify}(pk, \mathsf{b}, \mathsf{ID})$. Outputs 1 if $\mathsf{Hash}(pk, \mathsf{b}) = \mathsf{ID}$, 0 otherwise.

Reasonable values for n and m are 1024 and 257 bits respectively. Our tests were performed with those values. Note that the knowledge of a tuple (i, j, x_i, x_j) such that $g_i^{x_i} = g_j^{x_j}$ for some $i \neq j$ suffices to efficiently compute arbitrary hash collisions. Therefore, all nodes must trust the third party in charge of generating the global parameters, which must be chosen accordingly to [17].

For sake of brevity we do not provide a security proof for the hash algorithm. The security proof can be found in [17].

Probabilistic Batch Verification The homomorphic hash algorithm presented in [17] allows a node to verify many blocks probabilistically and in batches in order to improve efficiency. The idea of improving verification performances by batching is not new and appeared for the first time in [2]. Batch verification allows a peer to verify several block at once in order to improve aggregate verification speed. If the batch verifies successfully, then all blocks are correct with high probability, otherwise one or more blocks are corrupted. In order to perform a batch verification on a batch of size $\mathcal{B}$, a peer combines $\mathcal{B}$ blocks and their IDs using the $\mathsf{Combine}$ algorithm. Then it computes the homomorphic hash on the aggregate block and verifies whether it matches with the expected value obtained with the $\mathsf{Combine}$ algorithm.

Unfortunately, as pointed out by Gkantsidis and Rodriguez [13] and Yu et al. [26], an adversary can poison two or more blocks in a way that, when verified together in a batch, the resulting combined block is not affected by such modification. As an example, an adversary can corrupt blocks b_1 and b_2 by replacing them with $\mathsf{b}_1 + \varepsilon$ and $\mathsf{b}_2 - \varepsilon$. If the two blocks are verified separately, the verification would clearly fail, but if the receiver applies batch verification it will not detect the corruption. Both the cited papers provide a solution to this problem by multiplying each block by a random coefficient.

With this modification the adversary can only succeed by guessing the random coefficient correctly, which happens only with negligible probability.

We implemented the homomorphic hash algorithm to measure its performance on our test platform, performing both stand-alone verification and batch verification and compared the results. We based our implementation on the OpenSSL library (version 0.9.8g) under Linux. Our test were performed on a Intel Core 2 Duo T9500 at 2.6GHz. Table I shows the performances of our algorithm on the test machine. We performed batch verification for a batch of 1000 blocks. The size of each block is 16KB, and the size of batch window is therefore slightly less than 16MB.

TABLE I
PERFORMANCE EVALUATION OF THE HASH ALGORITHM. THE BATCH SIZE IS 1000 BLOCKS AND THE BLOCK SIZE IS 16KB.

Public parameter generation	1436 ms
Average time for calculating a hash	289 ms
Average speed for calculating a hash values	56 KB/s ($\sim$ 450Kbps)
Average time for composing 1000 hash values	2.9 ms
Average time for composing 1000 blocks	141 ms
Average time for verifying a batch	433 ms
Average speed for verifying a batch	37 MB/s ($\sim$ 300Mbps)

As our test confirmed, the cost of the composition of several blocks and their hash values is very low compared to the cost of calculating a single hash. Therefore the cost of verifying a batch is marginally higher than the cost of verifying a single block, and this holds even for rather large batches.

Batching clearly introduces a delay, since a newly received block cannot be used until a specific amount of data has been received and the whole batch verification is completed. How this delay affects network coding based systems was first studied by Gkantsidis and Rodriguez in [13]. They showed that a large batch window severely affects the performances of a content distribution network based on network coding, since explicit content requests pay a high delayed, proportional with the number of intermediate peers. This problem, of course, is mitigated when data traffic is mainly unidirectional. For instance, with content streaming or long downloads, the batching latency can be hidden after an initial "buffering" time; contents are then distributed in a pipeline fashion, and the batching does not increase the download time significantly.

B. Security Analysis

Given the adversary's compromise power and knowledge of the scheme, the scheme and usage mode in this paper effectively prevent an adversary from reaching its goal. By compromising a node, the adversary does not obtain any new secret information, since all nodes share the same information regarding packet verification. Moreover, even if the adversary subverts all but one nodes, the remaining legitimate peer will still be able to tell wether a packet has been polluted by any adversary-controlled node.

An adversary who has control over the generation of the global parameters for the homomorphic hash function can easily calculate collisions and therefore allow corrupt packets to propagate through the network. In this case a collisions implies that two or more blocks share the same block identifier. However, the **Parameter generation** constraint in our adversary model prevents this from happening. In practice this means that these global parameters must be chosen by a trusted third party. We point out that whenever the party which generates the global parameters misbehaves, legitimate peers can quickly notice thanks to the availability of hash collisions in the network. The peers can react by discard the global parameters and consider the third party as hostile. In order to be able to communicate again, they will have to find another trusted third party.

VII. BATCH VERIFICATION STRATEGIES

Batching is an optimistic approach in which verification performance is greatly improved when none of the blocks composing a batch is corrupted. Unfortunately if the verification of a batch fails, peers are left with no information about which block (or blocks) made the verification fail. Hence less efficient strategies must be applied. A peer can discard the whole batch without investigating further, but this approach does not make an efficient use of the resources since the bandwidth of the peer is limited and we expect some of the blocks to be corrupted. Therefore, when a batch fails a peer should try to keep as many non-corrupted blocks as possible to maximize bandwidth efficiency. A trivial way to find the non-corrupted blocks in a corrupted batch is to verify each block separately. This is very inefficient and has a cost which grows linearly with the number of blocks composing a batch. A more efficient strategy when the number of corrupted blocks is low is to perform a bisection search: the sub-batch is divided into two parts. Each part is verified as a new batch and is accepted if it is non-corrupted. The process is repeated on the part that does not verify correctly, until all corrupted blocks are found. When the number of corrupted blocks is small compared to the size of the batch this approach is quite efficient: it has a logarithmic complexity in the number of blocks composing a batch.

We studied how to provide a more efficient strategy for batch verification when a small number of blocks in a batch is corrupted. As stated in the Network Model (cf. section IV), we assume that the underlying communication layers provide reliable communication. Each peer is assumed to verify each block that is forwarded to other peers; therefore if a block is corrupted, it is so because of a malicious action, and the neighbor that forwarded it is the culprit of such corruption.

We assume that, since byzantine nodes can be detected as soon as they forward a corrupted packet, there will be a small number of them. If a batch does not verify correctly, we optimistically assume that only a subset of the neighbors are byzantine. To improve efficiency we propose the following *bisection* technique: when a batch fails, it gets divided it in sub-batches; each sub-batch is composed of all the blocks coming from one neighbor. Sub-batches which verify correctly can be accepted immediately; sub-batches which fail the verification can either be discarded completely or be verified with the bisection approach. However, discarding all blocks from a specific source may provide a good tradeoff between bandwidth efficiency and computational overhead. Once a node is discovered to be malicious, it can be removed from the neighbor list. This form of naive partitioning has a complexity which grows linearly with the number of neighbors.

A more efficient approach is to use the bisection technique at a sub-batch level instead of block level. We calculated the ratio between the expected number of verifications performed in the case of bisection and in the case of naive partitioning, varying the number of neighbors. The result is shown in Figure 1. When a node has a small number of neighbors, there is no much difference between the two approaches. The difference becomes more evident and justifies the adoption of this approach when the number of neighbors grow: when a node has eight neighbors, the expected number of verifications with neighbor-level bisection is 3/4 of the expected number of verification with naive partitioning. When the neighbors are sixteen or more, the bisection approach requires less than half verification compared to naive partitioning on average.

We have considered another batch verification approach which relies on statistics on the previous behavior of neighbors, called *ranked* partitioning. With this approach a node keeps track of how many blocks it received from each neighbor. In case a batch verification fails, the node divides the batches in sub-batches according to the source of the blocks, and orders them by the ranking of the source. Then it verifies the sub-batches sequentially, starting from the one with the lowest ranking. We argue that a neighbor with a high ranking is less likely to forward corrupted blocks than a neighbor with a low ranking. A denial of service attack against a peer executed by exploiting the batch verification algorithm has a higher effectiveness if it is performed by the neighbor with the highest ranking. However, in order to be among the neighbors with highest ranking, an adversary must first provide a large contribution in terms of non-corrupted blocks, therefore making the attack less effective (the user has to pay a higher computational cost to find the byzantine neighbor, but received a large number of non-corrupted blocks from that very neighbor) and particularly expensive to implement. Figure 2 shows the performances of this approach compared to the bisection methods. The graph represents the ratio of the expected values of the ranking partitioning and the bisection approach varying the number of neighbors. When a node has a very small number of neighbors, the ranked partitioning approach is less efficient than the bisection approach. Increasing the number of neighbors, the advantage of this batching method over bisection becomes more evident. A peer with sixteen nodes performs, on average, about half verification operations using the ranked partitioning approach compared to the bisection approach.

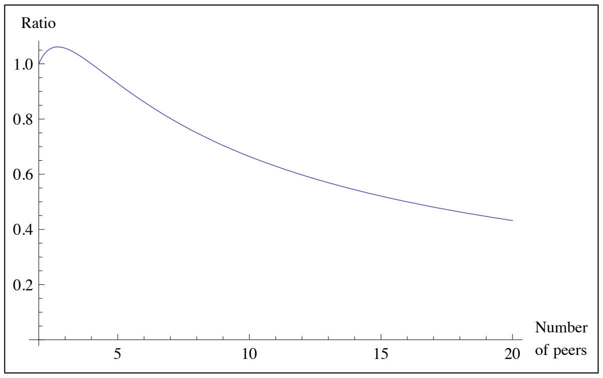

Fig. 1. Bisection vs. naive partitioning. The graph shows the ratio of the expected number of verification with bisection and naive partitioning.

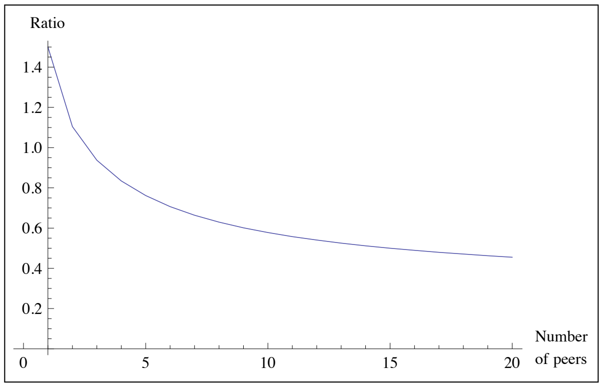

Fig. 2. Ranked partitioning vs. bisection. The graph shows the ratio of the expected number of verification with bisection and ranked partitioning.

VIII. Conclusions

The adoption of network coding in anonymous P2P systems for content distribution requires suitable schemes for integrity check aimed at mitigating pollution attacks while preserving the anonymity level. We have provided two novel contributions in this respect: (1) a modification of an existing integrity scheme [17], in the form of a different identification scheme for data blocks that requires neither a secure storage for hash values, nor a secure authenticated channel for retrieving such hash values; and (2) two strategies for checking the integrity of packets in batches, based on two different criteria for neighbor partitioning (*bisection* and *ranked*).

A small prototype which implements the integrity strategies described in this paper could show that the two batch verification strategies significantly boost the efficiency of integrity check under the realistic assumption that only a small fraction of neighbors are compromised. Performance measurements carried out on the prototype show that this solution is feasible for peers characterized by limited computing power.

References

[1] André Allavena, Alan Demers, and John E. Hopcroft. Correctness of a gossip based membership protocol. In *PODC '05: Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing*, pages 292–301, New York, NY, USA, 2005. ACM.

[2] Mihir Bellare, Juan A. Garay, and Tal Rabin. Fast batch verification for modular exponentiation and digital signatures. In *Advances in Cryptology – Eurocrypt 98*, pages 236–250. Springer-Verlag, 1998.

[3] Mihir Bellare and Phillip Rogaway. The exact security of digital signatures - how to sign with rsa and rabin. In *Advances in Cryptology - Eurocrypt 96*, pages 399–416. Springer-Verlag, 1996.

[4] K. Bennett and C. Grothoff. GAP: Practical Anonymous Networking. In *Proc. of Workshop on Privacy Enhancing Technologies (PET 2003)*, March 2003.

[5] Dan Boneh, David Freeman, Jonathan Katz, and Brent Waters. Signing a linear subspace: Signature schemes for network coding. In *Irvine: Proceedings of the 12th International Conference on Practice and Theory in Public Key Cryptography*, pages 68–87, Berlin, Heidelberg, 2009. Springer-Verlag.

[6] Philip A. Chou and Yunnan Wu. Network coding for the internet and wireless networks. Technical report, Microsoft Research, June 2007.

[7] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A Distributed Anonymous Information Storage and Retrieval S ystem. In *Proc. of Designing Privacy Enhancing Technologies: Workshop on D esign Issues in Anonymity and Unobservability (PET)*, pages 46–66, July 2000.

[8] R. Dingledine, M. J. Freedman, and D. Molnar. The Free Haven Project: Distributed Anonymous Storage Service. In H. Federrath, editor, *Proc. of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability (PET)*. Springer-Verlag, LNCS 2009, July 2000.

[9] R. Dingledine, N. Matthewson, and P. Syverson. Tor: The Second-Generation Onion Router. In *Proc. of the 13th USENIX Security Symp.*, San Diego, CA, August 2004.

[10] Taher E. Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proceedings of CRYPTO 84 on Advances in cryptology*, pages 10–18, New York, NY, USA, 1985. Springer-Verlag New York, Inc.

[11] R. Gennaro, J. Katz, H. Krawczyk, and T. Rabin. Secure Network Coding Over the Integers. In *Public Key Cryptography - PKC 2010, 13th International Conference on Practice and Theory in Public Key Cryptography*, 2010.

[12] C. Gkantsidis and P. Rodriguez. Network coding for large scale content distribution. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies, 13-17 March 2005, Miami, FL, USA*, pages 2235–2245, 2005.

[13] C. Gkantsidis and P. Rodriguez. Cooperative security for network coding file distribution. In *INFOCOM 2006. 25th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies*, 2006.

[14] C. Grothoff, K. Grothoff, T. Horozov, and J.T. Lindgren. An Encoding for Censorship-Resistant Sharing. Technical report, GNUnet project,http://gnunet.org/download/ecrs.pdf, 2005.

[15] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. Chapman & Hall/CRC, 2007.

[16] MinJi Kim, Muriel Médard, and João Barros. Counteracting byzantine adversaries with network coding: An overhead analysis. *CoRR*, 2008.

[17] 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*, pages 226–240, 2004.

[18] Jian Liang, Rakesh Kumar, Y. Xi, and Keith W. Ross. Pollution in p2p file sharing systems. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies, 13-17 March 2005, Miami, FL, USA*, pages 1174–1185, 2005.

[19] National Institute of Standards and Technology. *FIPS PUB 186-2: Digital Signature Standard (DSS)*. National Institute for Standards and Technology, Gaithersburg, MD, USA, January 2000.

[20] Marc Rennhard and Bernhard Plattner. Introducing MorphMix: Peer-to-Peer based Anonymous Internet Usage wi th Collusion Detection. In *Proc. of the Workshop on Privacy in the Electronic Society (WPES 2002)*, Washington, DC, USA, November 2002.

[21] Hans-Emil Skogh, Jonas Haeggstrom, Ali Ghodsi, and Rassul Ayani. Fast freenet: Improving freenet performance by preferential partition routing and file mesh propagation. In *CCGRID '06: Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid*, page 9, Washington, DC, USA, 2006. IEEE Computer Society.

[22] M. Waldman, A. Rubin, and L. Cranor. Publius: A Robust, Tamper-evident, Censorship-resistant and Source-anonymous Web Publishing System. In *Proc. of the 9th USENIX Security Symposium*, pages 59–72, August 2000.

[23] M. Wang and B. Li. How practical is network coding? In *Quality of Service - IWQoS 2006: 14th International Workshop, IWQoS 2006, New Haven, CT, USA, 19-21 June 2006, Proceedings*, pages 274–278, 2006.

[24] Yongge Wang. Insecure “provably secure network coding” and homomorphic authentication schemes for network coding. Cryptology ePrint Archive, Report 2010/060, 2010.

[25] B. Wilcox-O'Hearn. Experiences Deploying a Large-Scale Emergent Network. In *Proc. of the 1st Int.l Peer To Peer Systems Workshop (IPTPS02)*, March 2002.

[26] Zhen Yu, Yawen Wei, Bhuvaneswari Ramkumar, and Yong Guan. An efficient signature-based scheme for securing network coding against pollution attacks. In *INFOCOM 2008. 27th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies*, pages 1409–1417, 2008.