Securing coding-based cloud storage against pollution attacks

Cosimo Anglano, Rossano Gaeta, and Marco Grangetto, *Senior Member, IEEE*

Abstract—The widespread diffusion of distributed and cloud storage solutions has changed dramatically the way users, system designers, and service providers manage their data. Outsourcing data on remote storage provides indeed many advantages in terms of both capital and operational costs. The security of data outsourced to the cloud, however, still represents one of the major concerns for all stakeholders. *Pollution attacks*, whereby a set of malicious entities attempt to corrupt stored data, are one of the many risks that affect cloud data security.
In this paper we deal with pollution attacks in coding-based block-level cloud storage systems, i.e. systems that use linear codes to fragment, encode, and disperse virtual disk sectors across a set of storage nodes to achieve desired levels of redundancy, and to improve reliability and availability without sacrificing performance. Unfortunately, the effects of a pollution attack on linear coding can be disastrous, since a single polluted fragment can propagate pervasively in the decoding phase, thus hampering the whole sector.
In this work we show that, using rateless codes, we can design an early pollution detection algorithm able to spot the presence of an attack while fetching the data from cloud storage during the normal disk reading operations. The alarm triggers a procedure that locates the polluting nodes using the proposed detection mechanism along with statistical inference. The performance of the proposed solution is analyzed under several aspects using both analytical modelling and accurate simulation using real disk traces. Our results show that the proposed approach is very robust and is able to effectively isolate the polluters, even in harsh conditions, provided that enough data redundancy is used.

Index Terms—Cloud storage, coding, security, integrity, performance, pollution attack.

1 Introduction

Block-level cloud storage systems [1] provide the substrate allowing users and applications to attach their computing resources to remote, dynamically provisioned storage resources, that appear, behave, and can be used as local disks. Thanks to them, very large data sets can be stored without having to incur into potentially significant capital and operational expenses. However, to unlock their full potential, various problems need to be properly addressed, including performance of data access, as well as data availability and security.

Security of outsourced data to the cloud represents a key concern for users, system designers, and service providers. Among the many risks to data security, *pollution attacks* represent one of the most dangerous threats to data integrity, i.e., the ability of ensuring data trustworthiness. In this kind of attack, malicious entities take control of one or more storage resources to corrupt (*pollute*) data (or parts of it) so as to hinder data availability.

The negative impact of pollution attacks is further amplified when *coding techniques* are employed to represent data outsourced on storage resources. In this case, individual data items (each one stored independently from each other) are first subdivided in parts, that are then encoded to obtain a suitable number of *coded fragments* to be placed on a set of independent storage resources; the set of coded fragments must be computed such that a suitable subset of it allows the user to reconstruct the original data item. In this case, a couple of hard problems arise:

- in principle, any sequence of bits may be a valid coded fragment, so there is no simple mean to find out whether the data has been altered by a malicious storage node until the corresponding data item has been recovered by the user;
- even under the assumption that the above data item has been recovered, and it has been correctly detected as polluted, it is not trivial to understand which coded fragment(s) among those received by the user was polluted (and thus to identify the malicious storage resource responsible for that).

1.1 Our contribution

In this paper we propose a solution to both problems:

- we devise a *pollution detection algorithm* that detects, with high probability if a set of untrusted storage resources provides at least one polluted coded fragment. The algorithm is based on a modified version of the LT decoding algorithm exploiting Gaussian Elimination; since an analytical model for decoding (and detection) performance is unavailable in the literature we resort to simulations to estimate the detection probability.

• *Rossano Gaeta and Marco Grangetto are with Università degli Studi di Torino, Dipartimento di Informatica, Torino, Italia. Cosimo Anglano is with Università degli Studi del Piemonte Orientale, DiSIT-Computer Science Institute, Alessandria, Italia.*
E-mail: {first.last}@unito.it {first.last}@unipmn.it

- we design an *identification algorithm* that identifies the storage resources that are polluters with high probability. The algorithm we propose is not based on cryptographic checksums or digital signatures (hence it does not rely on the existence of a PKI or preestablished secure channels) and it only exploits coding redundancy and efficient decoding algorithms that require the solution of systems of linear equations.
- We perform an extensive evaluation of our algorithm using a combination of experimentation with a C++ prototype, analytical modeling and discrete-event simulations driven by real-word disk access traces. In particular, we assess its accuracy and time complexity, and we show that identification of malicious storage resources is possible with high probability and low running time for a wide range of coding redundancies. Moreover, we show that the average number of sector reads required to identify all polluters is very low and decreases as the coding redundancy increases.

We use the architecture of ENIGMA (defined in [2]) as a blueprint for the model of a typical cloud storage system based on LT codes, and we exploit some results reported [2] to set the values of various system parameters in the experimental evaluation. We would like to point out that in [2] we limited ourselves to quantify the ability of ENIGMA of merely *tolerate* the presence of polluters in the system, i.e. its ability of correctly reconstructing a sector assuming that a subset of its fragments have been altered, but we did not study the problem of detecting polluted sectors and of identifying malicious storage nodes responsible for that.

The paper is organized as follows. Sec. 2 discusses related works while in Sec. 3 we present the cloud storage model on which our work is based. Then, we continue with Sec. 4, where we discuss the attack model we consider in our work, and illustrate the pollution detection algorithm we devised. In Sec. 5 we move to the problem of identifying polluters, and we present our identification algorithm. In Sec. 6 we develop a mathematical model enabling us to study the time required to identify all polluters in the cloud storage system, that is validated against simulation results in Sec. 7. In this latter section, we also study, via experimentation and simulation, the accuracy vs. speed trade-off of the proposed algorithm. Finally, Sec. 8 concludes the paper, and outlines future research work.

2 Related work

Several papers have dealt with the problem of integrity check and repair of coding-based cloud storage systems. Closer to the spirit of our work are [3] and [4].

In [3] the authors consider random coding-based cloud storage and devise both a pollution detection algorithm and four identification and repair algorithms to recover the original data. The algorithms represent trade-offs between computational and communication complexity and successful identification (and repair) probability. This work differs from ours in many ways:

- the work in [3] exploits coding in $GF(q)$ with very large q to assume one extra coded fragment is enough to detect pollution, i.e., the pollution detection algorithm is assumed to be perfect. Conversely, we base our work on an imperfect pollution detection algorithm (see Alg. 1, Sec. 4) that stems from the use of LT codes based on simple XOR combinations ($q = 2$), and of small values of the coding block length k (that are preferred in the context we consider for the sake of performance and availability [2]). The imperfection of the pollution detection algorithms forced us to develop a more complex approach with respect to [3] because we simply cannot trust the (imperfect) detection mechanism to draw conclusions on the status of the system of equations. All algorithms in [3] would be more complex if an imperfect pollution detection mechanism had to be adopted;
- our pollution detection algorithm works incrementally as soon as an additional coded fragment is analyzed therefore it can detect pollution even before the data is recovered (it allows for a reduced running time). Conversely, the detection algorithm in [3] works right after the system of linear equations is solved. Alg. 1, being a variation of GE decoding follows the same principle of [3]. We provide the implementation details in the specific case of LT codes that, being suboptimal from the decoding overhead point of view, lead to suboptimal (imperfect) pollution detection;
- the output of the two polluter identification methods are different. Algorithms in [3] are invoked only if a random subset of cardinality $k + 1$ (out of n available equations) triggers the pollution detection, and aim to repair the data and to output a clean set of equations, from which the original data can be recovered. This means that only the polluted equations in the originally drawn subset are removed, while other polluted equations in the remaining $n - k - 1$ ones are left there. Our algorithm, instead, processes all the n equations at once and outputs the set of all malicious storage nodes. This involves a more complex organization of our method (Alg. 2, Sec. 5) and represents a significant difference with respect to algorithms in [3] (we believe that algorithms in [3] could well be adapted to output the entire set of polluted equations although to the price of a more complex structure;
- computational complexity of identification algorithms is sensitive to n and k since they are all based on trying all different subsets of equations until a clean one is found. In [5] the same authors devise a more efficient decoding algorithm that sometimes has to resort to a complex subspace search.

- besides characterizing the accuracy of our method (estimation of probability p_f) we conduct a more comprehensive analysis to evaluate the time required for identifying all polluters in the system.

In [4] rateless codes are exploited to devise a file based cloud storage system that achieves high availability and security; the paper mainly deals with data integrity and data repair and focuses on exact repair instead of a simpler functional repair of polluted coded fragments. The authors propose to use multiple LT encoding and decoding checks to avoid LT decoding failures; since the number of required encoding and decoding checks is equal to $\binom{n}{k}$, it follows that the data integrity check algorithm may become rather complex as the values of n and k increase. Nonetheless, only coding vectors are involved in the proposed approach that turns to be a one-time preprocess that can be reused on different files.

Cryptographic or algebraic based approached to design on-the-fly verification techniques of the received coded fragments is another line of research proposed and discussed by several papers, e.g., [6], [7], [8], [9], [10], [11], [12], [13], [14]. High computational costs for verification and remarkable communication overhead due to pre-distribution of verification information represent limitations of these approaches.

Besides verification, error correction of corrupted coded fragments is another important approach to deal with pollution attacks in coding-based systems, e.g., [15], [16], [17], [18]. All these methods are based on the addition of coding information that enable the coded fragment receivers to detect and automatically reconstruct the original data. The price to be paid is a remarkable increase in the coding overhead; furthermore, the effectiveness of these approaches heavily depends on the amount of corrupted information.

3 System model

The architecture of the cloud storage system we consider in this paper builds upon the ENIGMA distributed cloud storage infrastructure [2], that allows the provision of *Virtual Disks* (VDs), consisting of a set of consecutively-numerated sectors, that can be used as if they were standard physical disks. Its architecture features a set of N_S *Storage Nodes* (SNs), that store VD sectors after their proper encoding by means of *rateless codes*, and a *Proxy* where all the metadata allowing the retrieval and decoding of VD sectors are kept.

More precisely, ENIGMA uses Luby Transform (LT) rateless codes [19] to encode each sector, whereby each sector S is first split into k fragments of equal length $S = (s_1, \dots, s_k)$, from which n coded fragments $F = (f_1, \dots, f_n)$ are created [2]; these fragments are then placed on a subset $\mathcal{A}_S$ of the N_S storage nodes. The parameter k is known as *coding block length*, whereas n can be selected freely allowing to reach the desired level of redundancy n/k. After encoding, the n fragments of a given sector S are stored in group of x on a random

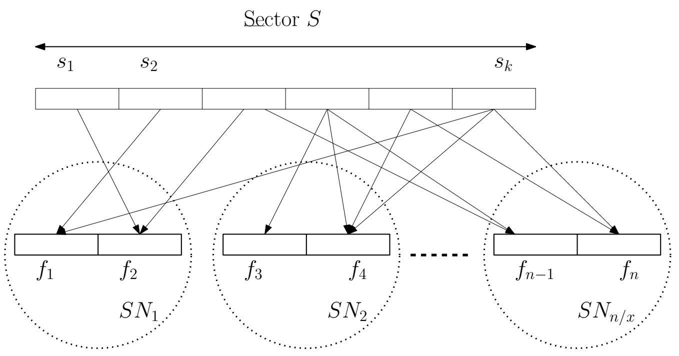

Fig. 1. Sector encoding and placement.

subset of the SNs. Thus, we have that every sector is stored on $|\mathcal{A}_S| = \lceil n/x \rceil$ different SNs.

In this paper we also assume that a subset of N_P of the N_S storage nodes are malicious and may intentionally corrupt the data they store (we call them *polluters*), and we also assume that the coded fragments of a given sector are stored by no more than n_P (out of the total N_P) polluters.

To read a sector S, all the SNs storing fragments of that sector are contacted by using the metadata stored on the Proxy. Upon receiving this request, each SN sends to the Proxy the x coded fragments of S it stores. The Proxy then progressively decodes the original sector S using the On-the-Fly Gaussian Elimination (OFG) [20], [21] algorithm. Any set of $k' \geq k$ coded fragments can be interpreted as a linear system of equations that can be solved with Gaussian Elimination to get the original k sector fragment (provided that k independent equations are available). The average number of fragments in excess $\epsilon = (k' - k)$ required for decoding is termed as coding *overhead*.

The overall scenario is graphically shown in Fig. 1 that represents a sector S first broken into k parts $S = (s_1, s_2, \dots s_k)$, and then placed on SNs in group of x coded fragments (in the pictorial representation we assume $x = 2$ for the sake of simplicity). The arrows are used to represent the contribution of each sector fragment to every coded fragments, and show that – as consequence of the encoding technique described in Sec. 4.1 – each coded fragment contributes to the decoding of several original sector fragments. For instance, the encoded fragments f_2 and f_4 can be used to decode two and three distinct sector fragments, respectively.

The dependency of several sector fragments from the same encoded fragment implies that, in case of pollution of the latter one, the decoding of all the former ones is potentially compromised, thus preventing the correct decoding of the corresponding sector. To increase robustness to pollution we impose a constraint on the placement of the coded fragments. In particular, we guarantee that the coded fragments referring to a given s_i are stored by at least $n_P + n_r$ SNs, where $n_r > 0$ can be tuned to obtain increased resilience to polluters by decreasing the probability that s_i is controlled only by malicious

TABLE 1
Paper notation.

S	VD sector
k	Number of sector fragments
n	Number of coded fragments
x	Grouping factor
s_i	i-th fragment sector
f_i	i-th coded fragment
N_S	Number of SNs
N_P	Number of polluters
n_P	Max. number of polluters storing a sector
n_r	Number of redundant SNs storing each s_i
$\mathcal{A}_S$	SNs storing sector S

SNs. Please note that n_r is not a free parameter since it must hold that $n_P + n_r \leq n/x$; therefore, higher values of n_r can be imposed by increasing the coding redundancy. It follows that the satisfaction of the placement constraint implies that at least $n_P + n_r$ coded fragments including s_i are available. Since encoding is quite efficient in this work we implement a simple rejection method where random placement of the coded fragments is iterated until the above constraint is fulfilled. Tab. 1 summarizes the notation used throughout the rest of the paper.

4 Attack model and pollution detection

In this section we model the pollution attack considered in this work, that consists in the injection of bogus coded fragments sent by malicious SNs in response to read requests, and we show how the OFG decoder (in charge of reconstructing the original sectors starting from the set of corresponding coded fragments) can be also used to carry out pollution detection.

4.1 Pollution Attack Model

The primary goal of polluters SNs is to make VD sectors unrecoverable by preventing the decoding of the original information while, at the same time, hiding their identity so as to make it difficult to recognize and remove them from the system. As already said, they attempt to achieve their goal by polluting in a certain fashion the coded fragment they store.

To explain how these pollution attacks may be carried out, we need to define how coded fragments are generated by the Proxy for given values on n, k, and x.

To simplify notation, and without loss of generality, in the following we assume that n is an integer multiple of x. Every coded fragment is computed as a XOR of a random set of sector fragments. More precisely, the i-th coded fragment of a given sector can be expressed as $f_i = \sum_{j=1}^{k} g_{i,j} s_j$ where we use the summation to represent the XOR operation, and $g_{i,j} = 1$ or $g_{i,j} = 0$ if the j-th fragment is included in the XOR or not, respectively.

The vector $g_i = (g_{i,1}, \ldots, g_{i,k})$ is known as the *coding vector*, and it is drawn randomly according to the procedure shown in [19], that guarantees optimal decoding properties. In particular, first a *degree* ρ is selected according to the so called Robust Soliton Distribution, and then a subset of ρ (out of k) sector fragments is randomly picked up for XOR. The coding vector is generated randomly and it is known only by the Proxy; from the point of view of SNs the coded fragments represent an unintelligible partial and randomized XORed segment of an unknown sector, thus guaranteeing privacy.

With this in mind, we can now define the type of pollution attacks we consider in this work. In particular, we assume that a polluter SN can reply to a read request by supplying a faked coded fragment f^p, created by XOR-ing an original coded fragment f with a random sequence $r \neq \mathbf{0}$ (where $\mathbf{0}$ denotes a string of 0s), i.e. $f^p = f + r$. In other words, a polluter transmits a fragment that is not in agreement with the coding vector known by data owner. Altering the coded fragment f to get f^p is a safe option for the polluter, since any receiving client that has not yet decoded the sector ignores the original information s_j and has no means to discriminate between polluted and non polluted fragments.

4.2 Pollution Detection

As shown in [2], rateless coding can be used to increase sector availability since the same original information can be retrieved from any random set of more than k coded fragments (the actual number of required fragments depends on the decoding overhead ϵ). At a first glance, it could seem that the use of coding can make the system very vulnerable to pollution. Indeed, as shown in Fig. 1, it is possible that a single polluted coded fragments propagates to many original sector fragments s_i. However, in this work we show that coding brings also significant benefits in terms of pollution detection, since it can be exploited to both detect pollution and identify the SNs responsible of the damage. To this end, we need to look in more detail at the LT decoding process.

LT decoding can be cast as the solution of a linear system of equations $GS = F$ where G is a $k \times k$ decoding matrix carrying on the rows k linearly independent coding vectors, F is the column vector of the k coded fragments corresponding to such coding vectors, and S is the vector of k unknown original sector fragments. In the following, we denote as G_l the l-th row of G and F_l the l-th element of vector F.

The OFG decoder [20] sequentially processes the input, i.e., the pairs (f_i, g_i), that are being provided by SNs, and executes Gaussian Elimination on the fly to progressively fill up G starting from an empty matrix, until G turns full rank, that corresponds to the recovery of the original sector.

A modified version of the OFG decoder, that includes the pollution detection mechanism we propose in this paper is shown in Alg. 1 below by using pseudo-code. The algorithm takes as input a set of coded fragments $\mathcal{Q}$, processes the coded fragments $(f, g) \in \mathcal{Q}$, and returns

Algorithm 1 Decode($\mathcal{Q}$)

```
1: $decoded =$ **false**, $polluted =$ **false**
2: **for all** $(f, g) \in \mathcal{Q}$ **do**
3:   **while true do**
4:     {Gaussian Elimination}
5:     $l \leftarrow$ position of leading one of $g$.
6:     **if** $G_l = \emptyset$ **then**
7:       $G_l \leftarrow g; F_l \leftarrow f$
8:       **break**
9:     **else**
10:      **if** $g = G_l$ **then**
11:        {$g$ is linearly dependent on current $G$}
12:        **if** $f \neq F_l$ **then**
13:          $polluted =$ **true**
14:        **end if**
15:        **break**;
16:      **else**
17:        {$g$ is linearly independent on current $G$}
18:        $g \leftarrow g + G_l$; $f \leftarrow f + F_l$
19:      **end if**
20:    **end if**
21:  **end while**
22: **end for**
23: **if** $G$ is full rank **then**
24:   $decoded =$ **true**
25: **end if**
```

two logic flags, namely *decoded* and *polluted*, indicating whether decoding is successful or not, and whether pollution is detected or not, respectively. It is worth recalling that here the goal is only to check whether the set $\mathcal{Q}$ includes or not polluted fragments and not to spot which ones have been actually modified.

OFG aims at selecting k linear independent equations (or linear combinations thereof) to fill the matrix G (initialized as an empty matrix). This is achieved by iteratively considering the position l of the leftmost 1 of every $g \in \mathcal{Q}$. If the l-th row of G is empty then g is copied to G_l while f is copied to F_l (see lines 5-8 in Alg. 1).

If, instead, G_l is already occupied, g is checked against it. If $g \neq G_l$ the algorithm performs a XOR operation with G_l, i.e. $g \leftarrow g + G_l$; $f \leftarrow f + F_l$ (lines 17-18 in Alg. 1) and the process is iterated on the resulting pair (f, g). Conversely, if $g = G_l$ (line 10 in Alg. 1), the equation is recognized as a linear combination of the current rows of G and in standard OFG decoder is simply discarded. In this paper we exploit this particular condition to perform pollution detection.

To this end, we note that in this case $g = \sum_{j=1}^{k} \alpha_j G_j$ and f can be computed according to the same combination of elements of F, i.e. $f = \sum_{j=1}^{k} \alpha_j F_j$.

From now on, let us assume we are processing a polluted fragment $f^p = f + r$. First of all let us recall that g corresponding to f^p is known only by the decoder and cannot be modified by the attacker; therefore, the OFG row insertion process will not be affected by pollution. When processing the pair (f^p, g) two outcomes are possible:

a) g is linearly independent on current G. In this case any polluted fragment is stored in a given row of G and F (lines 6-8 in Alg. 1);
b) g is linearly dependent on current G. In this case the decoder knows a linear combination $\sum_{j=1}^{k} \alpha_j F_j$ that should coincide with f^p. If all previous rows are clean, it turns out that $\sum_{j=1}^{k} \alpha_j F_j = f \neq f^p$ and therefore pollution is detected (lines 12-14 in Alg. 1). If polluted equations were inserted before (see previous item a) one gets $\sum_{j=1}^{k} \alpha_j F_j = f + \sum_{j=1}^{k} \alpha_j r_j$, that is a combination of random pollution patterns (assuming $r_j = \mathbf{0}$ for clean equations). It follows that, if the malicious SNs are not able to collude one another, it is likely to get $f + \sum_{j=1}^{k} \alpha_j r_j \neq f^p$ since $\sum_{j=1}^{k} \alpha_j r_j \neq r$ allowing one to detect pollution.

Finally, we note that the second check can detect pollution also when processing a clean equation, provided that at least a polluted one has already contributed to G according to a). In this case, the insertion of a clean equation g that is recognized as dependent on G will indeed allow the decoder to compute the check $\sum_{j=1}^{k} \alpha_j F_j = f + \sum_{j=1}^{k} \alpha_j r_j$ that has the chance to involve some polluted row with $r_j \neq \mathbf{0}$, that would violate the system of equations, thus revealing the presence of pollution.

Previous analysis shows that there is a chance to recognize pollution every time a linearly dependent equation is considered. It must be pointed out that the probability of such an event increases with the number $|\mathcal{Q}|$ of processed fragments. In fact, every new fragment progressively fills G, thus increasing the probability that a randomly encoded equation is redundant. This observation unveils that the more fragments the decoder can process (e.g., when a higher coding redundancy n/k is used), the more reliable the detection mechanism will be. Moreover, it must be noted that Alg. 1, being based on the observation of an inconsistency in the system of linear equations, cannot generate false pollution detections; in other words, our detector yields only false negatives and no false positives. In the following we denote as p_{det} the probability that Alg. 1 correctly detects pollution when at least one polluted fragment belongs to $\mathcal{Q}$.

5 Polluter identification algorithm

In this section we describe the algorithm we developed to identify the polluters in the system. We would like to stress that polluter identification is not an easy task, since the linear coding approach prevents the use of simple means to identify both which fragments have been altered and the SNs responsible for the damage.

In the following we consider a proxy that accesses a sector S using the retrieval procedure described in Sec. 3, and performs sector decoding by means of Alg. 1. If a pollution is detected by this algorithm, the Proxy triggers the polluters identification stage that aims at identifying – among the set A_S of SNs storing the n coded fragments of S – those SNs that have replied to the read request with polluted fragments.

During the polluters identification stage, the Proxy gathers all the n fragments corresponding to the sector that is found to be polluted, and uses them as input for the polluter identification algorithm; the sector is held at the Proxy (i.e. it is not forwarded to the requestor) during the identification. Conversely, those sectors that are found to be clean are handled as usual, that is once they have been recovered (using a suitable subset of the corresponding n fragments), they are passed to the users that requested them. It is worth pointing out that normal operations and polluter identification occur simultaneously.

In the remainder of this section, we first describe the polluter identification algorithm (Sec. 5.1), and then we discuss several building blocks that it leverages (Sec. 5.2) to carry out identification.

5.1 Identification algorithm

The polluter identification algorithm, described by Alg. 2, aims at computing the set of polluter SNs in $\mathcal{A}_S$ using an iterative approach. In each iteration, the algorithm analyzes a random subset of $\mathcal{A}_S$, and employs statistical inference to compute, for each SN, the probability of being a polluter. This probability drives the incremental computation of two sets, representing the polluters and the honest SNs, respectively. After each iteration the set of SNs yet to be analyzed shrinks until all SNs have been classified as either being honest or polluters.

Algorithm 2 Identify_polluters_set
```
1:  for $i = 1$ to $MaxAttempts$ do
2:    $\mathcal{U} = \mathcal{F} = \mathcal{L} = \mathcal{A}_S$
3:    $\mathcal{H} = \mathcal{P} = \emptyset$
4:    while ($|\mathcal{U}| > 0$ and ($\mathcal{F} \neq \emptyset$ or $\mathcal{L} \neq \emptyset$) and $|\mathcal{P}| \leq n_P$) do
5:      $\mathcal{F} = \mathcal{L} = \emptyset$
6:      for $j = 1$ to $MaxBP$ do
7:        $h = \min\{w-1, |\mathcal{H}|\}$
8:        $\mathcal{W} \subseteq^R_h \mathcal{H}$, $\mathcal{Y} \subseteq^R_{w-h} \mathcal{U}$
9:        $\mathcal{W} = \mathcal{W} \cup \mathcal{Y}$
10:       if (Decode($\mathcal{W}$).$polluted$) then
11:         $\{\mathcal{F}, \mathcal{L}\} \leftarrow$ BP_core($\mathcal{W}$,$\mathcal{U}$,$\mathcal{H}$)
12:         if ($\mathcal{F} \neq \emptyset$ or $\mathcal{L} \neq \emptyset$ ) then
13:           $\mathcal{P} = \mathcal{P} \cup \mathcal{F}$, $\mathcal{H} = \mathcal{H} \cup \mathcal{L}$, $\mathcal{U} = \mathcal{U} \setminus \mathcal{F}$, $\mathcal{U} = \mathcal{U} \setminus \mathcal{L}$
14:           break
15:         end if
16:       end if
17:     end for
18:     if (not Decode($\mathcal{H} \cup \mathcal{U}$).$polluted$) then
19:       $\mathcal{H} = \mathcal{H} \cup \mathcal{U}$
20:       $\mathcal{U} = \emptyset$
21:     end if
22:   end while
23:   if (not Sanity_checks($\mathcal{H}, \mathcal{P}$)) then
24:     $\mathcal{P} = \emptyset$
25:   else
26:     break {success: exiting from Algorithm.}
27:   end if
28: end for
29: return $\mathcal{P}$
```

Our algorithm uses the sets of SN identifiers defined below:

- $\mathcal{U}$, containing the identifiers of all the SNs whose state is still unknown;
- $\mathcal{P}$, containing the identifiers of the SNs that have been already classified as polluters;
- $\mathcal{H}$, containing the identifiers of the SNs that have been already classified as honest.

Moreover, we use the notation $A \subseteq^R_d B$ to denote that set A is a random subset of set B, with $|A| = d$.

Let us describe now the polluter identification algorithm, shown in Alg. 2. This algorithm uses a *divide et impera* approach to define several simpler identification problems. To this end it builds a working set $\mathcal{W} \subset \mathcal{A}_S$, of size $w < n$, by mixing h SNs randomly selected from $\mathcal{H}$ and the remaining $w - h$ from $\mathcal{U}$ (lines 7-9). At startup (lines 2-3), $\mathcal{H} = \emptyset$, so all the SNs are taken from $\mathcal{U} = \mathcal{A}_S$; as the identification proceeds, more and more (up to $w - 1$) honest SNs will be added to $\mathcal{W}$, thus easing the inference on the state of the SNs taken from $\mathcal{U}$ (only one in the most favorable case).

Then, the algorithm enters a loop (lines 4-22) in which – at each iteration – uses the *Decode* method (Alg. 1) to check whether the decoding of the fragments contributed only by nodes in $\mathcal{W}$ gives rise to a polluted sector (line 10). If this is the case, then at least one SN in $\mathcal{W}$ is a polluter. To identify these nodes, we resort to a statistical inference technique, known as *Belief Propagation* (BP), that has been already applied to the problem of polluter identification in the different scenario of coded peer-to-peer streaming [22].

In this paper, we apply BP to estimate the two SNs in $\mathcal{W}$ that are most likely to be polluter and honest (sets $\mathcal{F}$ and $\mathcal{L}$ at line 11, respectively) by calling the *BP_core* procedure, that is presented in Alg. 5. If *BP_core* succeeds (lines 12-15), then the memberships of sets $\mathcal{H}, \mathcal{P}, \mathcal{U}$ are updated and the **for** loop (lines 6 through 17) is exited. On the contrary, i.e. if either *BP_core* fails or if *Decode* does not detect pollution, then another attempt is made on a new random working set $\mathcal{W}$ up to a maximum number of trials ($MaxBP$).

When exiting the **for** loop (line 17) the sets $\mathcal{H}, \mathcal{P}$ have been possibly updated with the identification of a pair of SNs (polluter, honest). If we are able to decode from the set $\mathcal{H} \cup \mathcal{U}$, i.e. all the SNs identified so far plus all SNs still unknown, without detecting any pollution (line 18), then we can reliably assume all members of the union are honest; in such a case the algorithm exits the outer **while** loop.

The steps of the algorithm are iterated through the external **while** loop in order to progressively move SNs from $\mathcal{U}$ to $\mathcal{H}$ or $\mathcal{P}$ until one of the following conditions occur (**while** loop condition in line 4):

- there are no remaining SNs whose state is unknown, i.e. identification has been completed;
- *BP_core* failed all $MaxBP$ trials to identify either a polluter or an honest SN, i.e. the proposed decision metric does not allow to discriminate any further;
- the number of already identified polluters exceeds the maximum limit n_P.

The first case corresponds to a successful termination, whereas in the latter two cases both $\mathcal{H}$ and $\mathcal{P}$ are emptied to signal that the algorithm failed to identify the polluters.

BP_core, while being statistically solid, may fail identification when $p_{det} < 1$. Therefore, all the decisions taken on $\mathcal{F}$ and $\mathcal{L}$ cannot be considered as completely trustful. To avoid misclassifications, at the end of the **while** loop (line 23), sets $\mathcal{H}$ and $\mathcal{P}$ are tested using the *Sanity_checks* method defined in Alg. 6. The main identification algorithm successfully terminates by returning a non empty set $\mathcal{P}$ only if all sanity checks on $\mathcal{H}$ and $\mathcal{P}$ are passed (line 30). On the contrary, an additional attempt to compute $\mathcal{P}$ is performed up to a maximum number $MaxAttempts$. The repeated trials are useful since all attempts are driven by randomness, e.g., in the choice of working sets $\mathcal{W}$, that can lead to different outcomes.

To conclude, the identification algorithm can terminate either with success or with failure to compute the set of polluters $\mathcal{P}$. Therefore, it is characterized by the *failure probability* p_f that depends on all algorithm parameters that are summarized in Tab. 2. It is worth pointing out that a limited failure probability can be tolerated since during the normal operations carried on by clients on a VD several different sector reads can trigger the pollution detection and, as a consequence, different identification rounds are available. Therefore, doubtful identification outputs can be skipped waiting for a most favorable chance.

Algorithm 3 Build_random_factor_graph($\mathcal{W}$)

```
1: $\mathcal{A} = \mathcal{W}$
2: $\mathcal{C} = \mathcal{E} = \emptyset$
3: for $i = 1$ to $BP_w$ do
4:     $\mathcal{D}_i \subseteq_d^R \mathcal{W}$
5:     $c_i = (i, \text{Decode}(\mathcal{D}_i).polluted)$
6:     $\mathcal{C} = \mathcal{C} \cup \{c_i\}$
7:     for all $a \in D_i$ do
8:         $\mathcal{E} = \mathcal{E} \cup \{(a, c_i)\}$
9:     end for
10: end for
11: return $\mathcal{G} = (\mathcal{A}, \mathcal{C}, \mathcal{E})$
```

5.2 Building blocks

As discussed in the previous subsection, to carry out its operations Alg. 2 relies on two other procedures, namely *BP_core* and *Sanity_checks*, whose detailed descriptions follow.

5.2.1 Belief Propagation core algorithm

Polluter identification can be cast as a statistical inference problem as follows. The main idea is to characterize each SNs $i \in \mathcal{A}_S$ by an unknown (hidden) binary state χ_i, where $\chi_i = 1$ is used to identify a polluter and $\chi_i = 0$ is used to identify an honest SN. The goal is then to infer $\forall i \in \mathcal{A}_S, p(\chi_i = 1)$.

The probability distributions of $\{\chi_i\}$ are inferred by carrying out the following two phases:

1) first, we build a random instance of the so called *factor graph* $\mathcal{G} = (\mathcal{A}, \mathcal{C}, \mathcal{E})$ (see Alg. 3). The factor graph is a bipartite undirected graph where the first set of vertices ($\mathcal{A}$) represents SNs in $\mathcal{A}_S$, while the other one ($\mathcal{C}$) represents *checks*. The i-th check is represented by a pair of elements $c_i = (i, \text{Decode}(\mathcal{D}_i).polluted)$ where:
   - the first element (i) is a check identifier;
   - the second element is a boolean *check state* that is obtained by first computing a random subset $\mathcal{D}_i \subseteq \mathcal{A}$, and then by running the pollution detection and decoding algorithm (Alg. 1) on it. If the fragments in $\mathcal{D}_i$ lead to a clean decoding, then a *negative check* is created; conversely, if pollution is detected, then a *positive check* is created instead. We denote $|\mathcal{D}_i| = d$ as the *check size*.

It is worth pointing out that it is necessary to use BP with a decoding set $\mathcal{D}$ of size $d < w$ to have the chance to obtain negative checks, that represent the hints on which the identification of honest SNs is based. On the contrary, if we used $d = w$, then all the checks provided to BP would be positive, thus making identification impossible. The arcs of the factor graph $\mathcal{G}$ are created as follows: for each check $c_i \in \mathcal{C}$ the undirected arc $(a, c_i) \in \mathcal{E}$ if $a \in \mathcal{D}_i$, i.e. if the i-th check involves SN a.

2) next, we apply the *Belief_propagation* procedure to the factor graph $\mathcal{G}$ (see Alg.4) to obtain an estimate of the $p(\chi_i = 1)$ values given the current instance of $\mathcal{G}$. At startup we set $p(\chi_i = 1) = 0.5$ for all vertices in $\mathcal{A}$ not yet identified as honest. This value is meant to represent the maximum uncertainty with respect to the hidden state of SNs. On the contrary, all SN in $\mathcal{H}$ have their $p(\chi_i = 1)$ probabilities set to 0, i.e., honest SNs are polluters with zero probability.
Subsequently, the BP algorithm is run on the factor graph to obtain an estimate of the $p(\chi_i = 1)$ values. The mathematical details and approximations required to run BP along the bipartite graph of SNs and checks can be found in [22]. Nonetheless, Alg. 4 summarizes the key steps; intuitively, a negative check c_i contributes to lower the probability of SNs in $\mathcal{D}_i$ to be malicious while a positive check would increase it. To counteract the fact that the checks are not fully reliable since $p_{det} < 1$, several runs of the BP inference are used on different random subsets $\mathcal{D}_i$ and the estimated probabilities are accumulated. As shown in Alg. 4 the BP inference algorithm (referred to in the pseudo-code as an external function named *BP_inference* ($\mathcal{G}$)) is used BP_t times, and the output estimates $p(\chi_i = 1)$ are summed onto $P(\chi_i)$, that will represent the decision metric used for identification. At the end of the cycle, $P(\chi_i)$ is the average probability of being a polluter, estimated on BP_t different instances of $\mathcal{G}$. It can be noted that the computation of the $P(\chi_i)$ is based on several checks and factor graph instances and therefore the negative effect of the unreliable pollution detection mechanism (that could erroneously assign the check state) is attenuated when the probability of false negatives in Alg. 1 is low.

Algorithm 4 Belief_propagation($\mathcal{W}, \mathcal{H}$)
```
1: for all $a \in \mathcal{W}$ do
2:     $P(\chi_a) = 0$
3:     if $a \in \mathcal{H}$ then
4:         $p(\chi_a = 1) = 0$
5:     else
6:         $p(\chi_a = 1) = 0.5$
7:     end if
8: end for
9: for $i = 1$ to $BP_t$ do
10:    $\mathcal{G} \leftarrow$ Build_random_factor_graph($\mathcal{W}$)
11:    $\{p(\chi_a)\} \leftarrow$ BP_inference($\mathcal{G}$)
12:    for all $a \in \mathcal{W}$ do
13:        $P(\chi_a) = P(\chi_a) + p(\chi_a = 1)$
14:    end for
15: end for
16: for all $a \in \mathcal{W}$ do
17:    $P(\chi_a) = P(\chi_a)/BP_t$
18: end for
19: return $\{P(\chi_a)\}$
```

Alg. 5 describes the *BP_core* method that exploits the two previous algorithms to create our core decision method based on the output of BP. This algorithm shows how to threshold $P(\chi_i)$ to decide on the most likely polluter and honest node, respectively. In particular, SN f is inserted in set $\mathcal{F}$ (containing the top suspect SN) only if $P(\chi_f)$ exceeds the threshold η_f. A similar reasoning is carried out to assign an element to set $\mathcal{L}$ that contains the identity of the SN that is most likely to be honest (l).

The **if** statement at line 3 discriminates the behavior of the algorithm depending on the number of SNs whose state is still unknown: if such a number is larger than k/x, we found that is convenient to use the probability ranking to discriminate between the most and least likely to be polluters and both f and l SNs can be identified; otherwise only the SN f with the largest $P(\chi_f)$ is identified (in this case the number of SNs in ranking is limited and we do not assume the least likely being a honest one). This sets will then be used to progressively refine the identification of all the malicious nodes forming the sets $\mathcal{H}$ and $\mathcal{P}$ that have been defined previously.

5.2.2 Sanity checks

Since all phases of our method are based on an unreliable pollution detection mechanism (as discussed in Sec. 4.2, Alg. 1 may yield false negatives with probability p_{det}), to avoid misclassifications we also apply sanity checks algorithms to the output of the identification phase, i.e., sets $\mathcal{H}$ and $\mathcal{P}$. To this end, Alg. 6 verifies the following constraints:

- the SNs in $\mathcal{H}$ allow decoding of the sector without pollution (line 2);
- the SNs in $\mathcal{P}$ are actual polluters (**for** loop in lines 3-14). This check is performed by decoding a working set $\mathcal{Y}$, composed of one polluter from $\mathcal{P}$ and all but one honest node from $\mathcal{H}$ to verify that pollution actually occurs. This check is carried on all possible elements in $\mathcal{P}$ and $\mathcal{H}$. The check fails as soon as a clean decoding is detected.

Algorithm 5 BP_core($\mathcal{W}, \mathcal{U}, \mathcal{H}$)
```
1: $\{P(\chi_a)\} \leftarrow$ Belief_propagation($\mathcal{W}, \mathcal{H}$)
2: $f = \arg\max_a\{P(\chi_a)\}$, $l = \arg\min_a\{P(\chi_a)\}$
3: if ($|\mathcal{W} \cap \mathcal{U}| > k/x$) then
4:     if ($P(\chi_f = 1) \geq \eta_f$ and $P(\chi_l = 1) \leq \eta_l$) then
5:         $\mathcal{F} = \{f\}$, $\mathcal{L} = \{l\}$
6:     end if
7: else
8:     if ($P(\chi_f = 1) \geq \eta_m$) then
9:         $\mathcal{F} = \{f\}$
10:    end if
11: end if
12: return $\{\mathcal{F}, \mathcal{L}\}$
```

Algorithm 6 Sanity_checks($\mathcal{H}, \mathcal{P}$)

```
1: $success$ = true
2: if (Decode($\mathcal{H}$).$decoded$ and not Decode($\mathcal{H}$).$polluted$) then
3:     for $p \in \mathcal{P}$ do
4:         for $h \in \mathcal{H}$ do
5:             $\mathcal{Y} = \mathcal{H} \cup \{p\}$, $\mathcal{Y} = \mathcal{H} \setminus \{h\}$,
6:             if (not Decode($\mathcal{Y}$).$polluted$) then
7:                 $success$ = false
8:                 break
9:             end if
10:        end for
11:        if (not $success$) then
12:            break
13:        end if
14:    end for
15:    if ($success$) then
16:        if (decoding of $\mathcal{H}$ depends on single equation) then
17:            $success$ = false
18:        end if
19:    end if
20: else
21:    $success$ = false
22: end if
23: return $success$
```

- sector decoding does not depend on a single equation since in this case there is clearly no way to verify whether such equation is polluted or not. Of course, this check potentially classifies as failed a correct identification attempt.

5.3 Failure probability vs. speed trade-off

Clearly, the failure probability p_f of Alg. 2 depends on all its parameters and a trade-off arises between the algorithm speed and the p_f values. In particular, this trade-off is determined by the:

- robustness of the BP based inference that increases as the overall number of checks in the factor graphs we randomly generate increases. Of course, the larger the factor graph the slower Alg. 4. The size of the factor graphs is given by $BP_t \cdot BP_w$;
- thresholds η_f, η_l, and η_m in Alg. 5 determine the accuracy required to identify one polluter and a potential honest SN. Tight thresholds require to generate a larger number of random factor graphs before reliable identification (i.e., early exiting the **for** loop in lines 6-17 of Alg. 2). Parameter $MaxBP$ represents an upper bound to the number of such attempts.
- the maximum number of identification trials ($MaxAttempts$). Indeed, identification is repeatedly attempted to take advantage of randomness both in the choice of working sets $\mathcal{W}$ and in the creation of checks in the factor graphs. Clearly, the larger $MaxAttempts$ the higher the probability to run a successful identification attempt at the expense of increasing the algorithm execution time.

6 Mathematical model

In this section, we develop a mathematical model to represent the average number of sector requests to identify all N_P polluters among the N_S storage nodes. To this end, let us define a *trace* of disk sector reads as a sequence $\{S(t)\}$ where $S(t)$ represents the t-th disk sector request.

Every sector read triggers the decoding and detection Alg. 1; in case pollution is detected, Alg. 2 is invoked to attempt identification. We then observe that the N_S SNs of the cloud storage system are progressively partitioned into three disjoint subsets: identified polluters, unidentified polluters, and honest.

We denote the number of identified polluters as $0 \le N_I \le N_P$ and the number of unidentified polluters as $0 \le N_U \le N_P$, with the constraint that $N_I + N_U = N_P$. It follows that the number of honest nodes is equal to $N_S - N_I - N_U$.

The allocation of the n fragments of a sector S to $\frac{n}{x}$ SNs can be viewed as just as many draws without replacement from all the SNs. In this case, it is well-known that the probability a sector has been allocated to n_I identified polluter and to n_U unidentified polluters (hence to $\frac{n}{x} - n_I - n_U$ honest storage nodes) follows a multivariate hypergeometric distribution

$$h(n_I, n_U, N_I, N_U) = \frac{\binom{N_I}{n_I}\binom{N_U}{n_U}\binom{N_S - N_I - N_U}{\frac{n}{x} - n_I - n_U}}{\binom{N_S}{\frac{n}{x}}} \qquad (2)$$

where h is equal to 0 if $n_I + n_U > \frac{n}{x}$. We denote as $p_{id}(t, y)$ the probability that after processing the request for sector $S(t)$ exactly $0 \le y \le N_P$ polluters have been identified. We can express $p_{id}(t, y)$ recursively as in Eq. 1 where $l = \min(\frac{n}{x}, N_P)$.

The base case (for $S(1)$) refers to the very first sector processing; the number of identified polluters remains equal to 0 either if the sector has been entirely allocated to honest storage nodes (first term) or if Alg. 2 fails when u unidentified polluters store fragments. On the other hand, y polluters are identified only if the sector has been allocated to y unidentified polluters (factor $h(0, y, 0, N_P)$) and if Alg. 2 is successful on y polluters (factor $(1 - p_f(y))$).

The general expression (for $t > 1$) is the contribution of three cases:

- the first case states that y polluters are identified when processing sector $S(t)$ if $y - u$ were identified at the previous step (request for sector $S(t-1)$) and Alg. 2 is successful on a sector allocated to u unidentified polluters and to i already identified ones. All the possible cases are taken into account by summing over all feasible values for u and i;
- the second case considers when 0 unidentified polluters store fragments of $S(t)$ for all feasible values of already identified ones;
- the third case represents the situation when $S(t)$ is allocated to i identified and to u unidentified polluters but Alg. 2 fails. All feasible cases for i and u are taken into account by the double summation.

Eq. 1 represents a discrete time model that describes the number of identified polluters. The t^{th} time step of the model represents the processing of $S(t)$ by Alg. 2. We can thus start from Eq. 1 to define the average time (number of sector read requests) to identify all N_P polluters as

$$n_{clean} = \sum_{t=1}^{\infty} t \cdot [p_{id}(t, N_P) - p_{id}(t-1, N_P)] \qquad (3)$$

where we assumed that $\forall i, p_{id}(0, i) = 0$.

7 Experimental results

In this section we provide results characterizing our proposal. The section is organized in three parts:

1) in Sec. 7.1 we define the parameters setting for Alg. 2. To this end, we first analyze the performance of the pollution detection algorithm (Alg. 1) in terms of probability p_{det}; this preliminary step allows us to set the values of parameters w and d required by Alg. 2 to identify polluters;
2) in Sec. 7.2 we provide a sensitivity analysis of Alg. 2 in terms of probability p_f as a function of several parameters;
3) in Sec. 7.3 we consider a complete cloud storage system and we evaluate the time required to expunge *all* polluters from the system.

The first two parts of the analysis are carried out with a C++ prototype implementing all algorithms described in Sec. 4 and 5. Probabilities p_{det} and p_f are estimated by running 1,000,000 trials. In the third case, we develop a trace-driven, discrete event simulator, that we use to reproduce the dynamic behavior of the whole system by considering requests to disk sectors extracted from real traces; we also exploit the numerical solution of the mathematical model developed in Sec. 6 that is validated against simulation results.

Unless otherwise stated, all results are worked out with the coding and placement parameter $k = 32$, $n = 2k$, $x = 4$, that in [2] have shown to yield an optimum compromise among the different VD properties and performance when using a small redundancy factor, i.e., $n = 2k$.

The parameter n_P can be chosen as a function of the LT coding parameters. In particular, we impose that the honest SNs are enough to permit LT decoding, i.e. there are at least k'/x honest SNs; therefore we set $n_P = \frac{n-k'}{x} = \frac{n-k-\epsilon}{x}$. In our settings we use $\epsilon = 12$: in this case the probability to successfully decode from $(k+\epsilon)/x$ SNs (with Alg. 1) turns to be 0.9926. Furthermore, we set $n_r = 2$, i.e. we assume that each sector fragment is dispersed on at least $n_P + 2$ SNs.

7.1 Parameters setting

The first step of our analysis is to characterize the capability of correctly spotting a sector as polluted, i.e., to evaluate the detection probability p_{det} of *Decode* (Alg. 1).

Our C++ prototype implements two different attack models: in the first case (*type A* attack) a polluter modifies *all* the x fragments it holds, whereas in the second case (*type B* attack) it modifies *only one* fragment out of x, instead. In Fig. 2 p_{det} is shown as a function of $|\mathcal{Q}|$ (the SNs set size used for decoding) for several values of the number of polluters $m \leq n_P$ in the sector. The left plot refers to type A attack whereas the right one represents type B attack.

As discussed in Sec. 4.2, it can be noted that p_{det} increases with $|\mathcal{Q}|$ up to maximum reliability, i.e., $p_{det} = 1$. Moreover, it can be observed that p_{det} increases also with m; in this case, it is more likely that multiple polluted coded fragments trigger the consistency check in Alg. 1. Finally, it is worth pointing out that type B attack yields lower values of p_{det} w.r.t. type A attack, i.e., it makes pollution detection less reliable. For example, when $|\mathcal{Q}| = 9$ and $m = 1$ we obtain $p_{det} = 0.9998$ for type A attack and $p_{det} = 0.9206$ for type B attack.

We exploit the results for p_{det} to set the values of two key parameters of Alg. 2:

- the working set size w must be chosen so as that any subset including at least one polluter is reliably detected. As shown in Fig. 2 (where $|\mathcal{Q}|$ on the x-axis must be interpreted as w), the larger w the more reliable the detection. Therefore, we set $w = 13$, that yields $p_{det} \geq 0.99999$ in our worst case attack scenario (type B attack in Fig. 2).
- the decoding set size $d \leq w$ must be chosen as a tradeoff between the probability to have a small subset to decide upon, and the reliability of detection. To strike a balance between the opposite needs we use $d = 10$ that yields $p_{det} = 0.99374$ in the worst

TABLE 2
Algorithms parameters

Parameter	Algorithm	Value
w	2	13
$MaxBP$	2	10,25,100
$MaxAttempts$	2	1-5
d	3	10
BP_w	3	w
BP_t	4	7
η_f	5	0.6
η_l	5	0.4
η_m	5	0.8

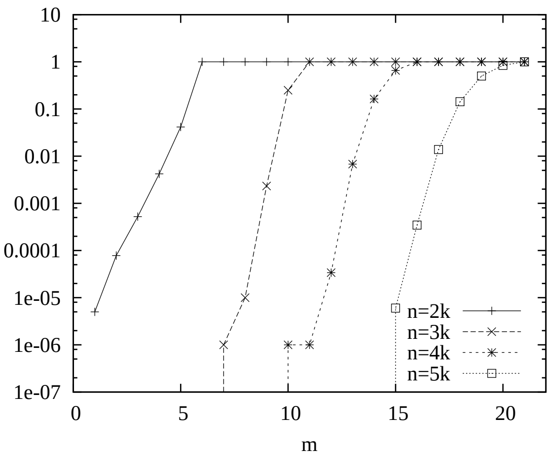

Fig. 4. Identification failure p_f as a function of the number m of malicious SNs for several values of n.

case shown in Fig. 2, where $|\mathcal{Q}|$ on the x-axis must be interpreted as d now.

Finally, taking into account the analysis of the BP algorithm worked out in [22], the BP inference window BP_w has been given the same size of the working set w, while the number of BP rounds BP_t has been set to the value of 7. All remaining parameter values are summarized in Tab. 2.

7.2 Sensitivity analysis

Here we analyze the performance of Alg. 2 with the parameters setting we defined in the previous section to consider the effect of parameters $MaxBP$. In Fig. 3 (left) p_f is shown as a function of m when increasing $MaxBP$ from 10 to 100. It can be noted that $MaxBP$ can be used to significantly boost the identification algorithm performance in particular when $m = n_P$, i.e. when coping with the maximum number of tolerable polluters in a single sector. As an example, setting $MaxBP = 100$ reduces p_f by almost two orders of magnitude with respect to the case $MaxBP = 10$.

A similar behavior is obtained when fixing $MaxBP = 100$ and considering different values for parameter $MaxAttempt$: in Fig. 3 (right) p_f is shown for some values of the parameter in the range from 1 to 5. Also in this case the gain is significant in the case $m = 5$ for $MaxAttempt = 5$.

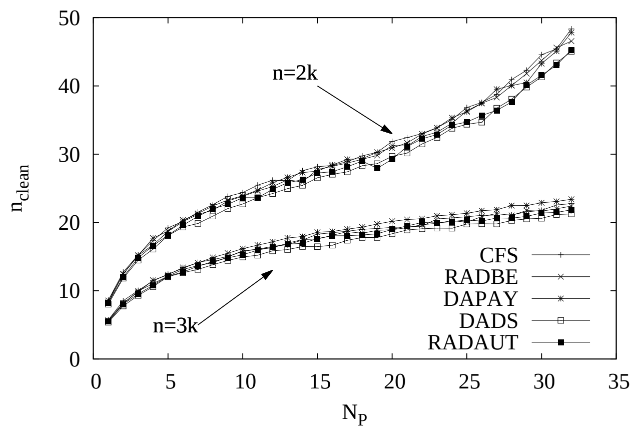

Fig. 5. n_{clean} values computed by the simulator for $n = 2k$ and $n = 3k$.

Now we set $MaxBP = 100$, $MaxAttempt = 5$ and we analyze p_f as function of the coding redundancy. In Fig. 4 p_f is shown as function of m when $n = 2k, 3k, 4k, 5k$. It can be noted that the algorithm we designed is able to exploit the coding redundancy to identify an increasing number of malicious SNs in a single sector. As an example, if one targets $p_f \leq 10^{-3}$ it can be observed from the reported results that 3, 7, 12 and 16 polluters can be identified out of 16,24,32 and 40 SNs; in other words the percentage of identified malicious SNs in a single sector increases from 18% ($n = 64$) up to about 40% ($n = 160$).

7.3 Performance Analysis

In this section we evaluate the performance of the identification algorithm in a complete framework using the following metrics:

- n_{clean} as defined in Eq. 3, that is the number of sectors processed by the identification algorithm before all polluters are identified;
- t_{clean}, that is the amount of time elapsed from the beginning of the experiment until all polluter are identified (to assess the performance of the algorithm when all timings, e.g., inter-arrival times of sector requests, are taken into consideration).

While, from a theoretical standpoint, n_{clean} allows us to assess the complexity of the polluter detection algorithm, from the perspective of a user the actual time t_{clean} taken to clean the system is more relevant.

The mathematical model discussed in Sec. 6 is used to compute n_{clean}, while t_{clean} is estimated by means of a discrete-event trace-driven simulator. This simulator is also used to validate the above mathematical model.

7.3.1 System simulator

The simulator we developed takes as input a description of an ENIGMA configuration, expressed in terms of its relevant system parameters, and a trace of disk sector