Malicious Node Identification in Coded Distributed Storage Systems under Pollution Attacks

Abstract

In coding-based distributed storage systems (DSS) a set of storage nodes (SN) hold coded fragments of a data unit that collectively allow one to recover the original information. It is well known that data modification (a.k.a. pollution attack) is the Achilles' heel of such coding systems; indeed, intentional modification of a single coded fragment has the potential to prevent the reconstruction of the original information because of error propagation induced by the decoding algorithm. The challenge we take in this work is to devise an algorithm to identify polluted coded fragments within the set encoding a data unit and to characterize its performance.

To this end, we provide the following contributions: i) we devise MIND (Malicious node IdeNtification in DSS), an algorithm that is general with respect to the encoding mechanism chosen for the DSS, it is able to cope with a heterogeneous allocation of coded fragments to SNs, and it is effective in successfully identifying polluted coded fragments in a low-redundancy scenario; ii) we formally prove both MIND termination and correctness; iii) we derive an accurate analytical characterization of MIND performance (hit probability and complexity); iv) we develop a C++ prototype that implements MIND to validate the performance predictions of the analytical model.

Finally, to show applicability of our work, we define performance and robustness metrics for an allocation of coded fragments to SNs and we apply the results of the analytical characterization of MIND performance to select coded fragments allocations yielding robustness to collusion as well as the highest probability to identify actual attackers.

Malicious node identification in coded distributed storage systems under pollution attacks

Rossano Gaeta, Marco Grangetto

March 11, 2022

1 Introduction

Distributed storage systems (DSS) are employed to reliably hold data units by means of a set of storage nodes (SN). As opposed to traditional monolithic enterprise storage, DSS can achieve better scalability, load adaptation, geographical migration and fault tolerance, just to name a few advantages. DSS have found their way in both research and commercial scenarios including cloud storage data centers, peer-to-peer systems, wireless sensor networks, fog/edge computing systems [1, 2, 3, 4, 5, 6, 7]. Nonetheless, reliability and security of DSS are still concerns that can significantly limit their adoption [8, 9].

In this context, coding based DSS have been proposed to improve many aspects of the system, namely increasing throughput, reducing latency, simplifying data collection, and granting reliability [3, 10, 11, 12]. Coding redundancy of DSS represents a key ingredient to ensure data units' reliability and availability. Redundancy can be realized by means of *coding* techniques whereby a data unit is divided into k fragments and n coded fragments are produced by some suitable linear erasure code. Then, the original data units can be recovered from a set of $q \geq k$ coded fragments. In this work we are going to show that redundancy can also be exploited to secure the system against intentional or unintentional data modification.

Most of the designs and analysis of coding-based DSS have been developed under the (sometimes implicit) hypothesis that SNs are homogeneous [13], e.g., the responding probability is the same for all SNs. As a consequence of the homogeneity assumption, uniform allocation of coded fragments to SNs represents the optimal setting from both the theoretical and practical point of views. Nevertheless, many (if not all) systems where DSS have found their way are heterogeneous in nature, e.g., peer-to-peer systems and fog/edge computing. This is true also in more controlled scenarios such as cloud storage data centers; indeed, components are periodically being upgraded and replaced leading to the co-existence of different device models with diverse storage capacities, expected lifetimes, access speed, and so on. Only recently, some research studies focused on the problem of allocating coded fragments to *heterogeneous* SNs, i.e., charac-

terized by different reliability levels, to maximize the probability of successfully retrieving the original data [14, 15].

In general, coding-based DSS have proved to be effective in delivering high levels of reliability, availability, and performance provided that high enough redundancy can be deployed, i.e., when the number of coded fragments n is several times the number of data fragments k. The need for an efficient resource management, e.g., in cloud storage data centers, or an intrinsic characteristic of the system application, e.g., peer-to-peer based live streaming of multimedia context, in this kind of systems has driven another line of research that deals with the design of *low-redundancy* coding techniques, i.e., when $n \leq 2k$.

Nevertheless, it is well known that data modification (a.k.a. *pollution attack*) is the Achilles' heel of coding systems [16]. Indeed, intentional modification of a single coded fragment has the potential to propagate its bogus effects during the decoding process to such an extent that the original data is completely damaged. DSS can efficiently resist and react to pollution attacks although existing methods have been devised under the assumptions of high redundancy and in the case of homogeneous storage, i.e., each SN is given the same amount of coded fragments representing a data unit. A pollution attack in a low redundancy scenario is even more critical, instead, because the number of coded fragments that can be lost (modified) without affecting data availability is very limited.

Checksum recomputing and comparison in all those contexts wherein for each of the n coded fragment a checksum can be computed, stored, and communicated via a trusted infrastructure to clients is a viable and effective solution to resist to pollution attacks and to identify malicious SNs. Nonetheless, in many other contexts, e.g. in completely decentralized systems such as peer-to-peer systems, wireless sensor networks, and fog/edge computing systems, the access rate to data units, the constraints on meta-data storage and communication make the solution based on checksums either unfeasible or too resource demanding. In these cases, alternative solutions should be developed.

Our contribution

The goal of the paper is to devise an accurate, efficient, and robust alternative identification of malicious SNs that does not rely on meta-data exploitation and that is solely based on decoding. Furthermore, the technique should not rely on restrictive assumptions both on coding mechanism and on fragments allocation to SNs and has to be amenable to formal characterization of its correctness and performance. The output of the algorithm is a list of sets of coded fragments containing polluted ones along with the identity of the actual SNs holding them.

To this end, we focus on the problem of polluter attacks in low-redundancy, coding-based, heterogeneous DSS. We provide contributions in terms of algorithmic aspects of the identification method and discuss its correctness. We also consider its performance by deriving analytical bounds on the hit probability and complexity of the proposed technique. Our contributions are as follows:

- we devise MIND, an algorithm to identify polluted coded fragments that is built around the simple idea to progressively isolate sets of coded fragments that do not trigger a pollution detection mechanism. The algorithm input is a possibly heterogeneous allocation of coded fragments to SNs and its output is a set of coded fragments that contain all polluted ones.
- We discuss the MIND termination and correctness; we formally prove that when MIND returns a solution it is correct with a probability that can be made arbitrarily close to 1 by tuning coding parameters.
- Since MIND is probabilistic we derive an analytical characterization of its hit probability and complexity as a function of the parameters of the algorithm, the decoding probability, and the DSS settings.
- We developed a C++ prototype that implements MIND to validate the predictions of the analytical model we derive.

Some remarks

Although MIND is an *extremely* simple algorithm its proof of correctness and the analytical characterization of its performance *are not*. Nevertheless, MIND simplicity makes it an ideal candidate for deployment in high throughput environments where accuracy and complexity have to be traded-off to meet the desired design goal.

It is worth pointing out that MIND and our mathematical modeling of its performance and complexity are quite general and can be both applied to any fragment encoding technique provided that a pollution detection mechanism is available. To show the effectiveness of our proposal, in this paper we consider Random Linear Network Codes (RLNC) whose decoding probability is analytically characterized in [17].

As an example of how MIND could be exploited in a cloud storage system, in Section 6 we define the performance of an allocation by using two indexes: the *hit probability* that represents how likely MIND is able to successfully terminate, and the *collusion resistance* that is an indicator function to represent if an allocation is able to resist collusion among malicious SNs. Based on these results we select the allocations achieving both robustness to collusion as well as the highest polluter identification accuracy using MIND.

Unfortunately, the design of allocation of coded fragments to SNs in presence of pollution attacks for low-redundancy, coding-based, heterogeneous DSS is an unclear issue that received little or no attention so far from the scientific community. In particular while some works have focused on optimal allocation to cope with unreliable SNs we are not aware of studies analyzing some key security aspects such as SNs collusion potential in a heterogeneous scenario. Nevertheless, the goal of this part of our paper is to provide an example of how the analytical characterization of MIND performance could be exploited to support the design of a realistic scenario. The design of efficient algorithms to solve the optimization problems defined in Section 6 is outside the scope of

this paper and would deserve an intensive research effort per se. Therefore, we used a straightforward exhaustive search for the optimal solutions; despite this inefficient approach to the solution of our optimization problems all results have been computed once for all in just a few hours on an i9 CPU.

Paper organization

The paper is organized as follows: Section 2 describes the coding-based DSS we consider as well as the attack model we deal with. Section 3 contains the detailed MIND algorithm specification by means of pseudo-code as well as formal proof of termination and correctness. Section 4 illustrates the derivation of the analytical characterization of MIND performance and robustness whose validation is carried out in Section 5. Section 6 comments on the results obtained from the search for optimal allocations in a wide range of values of system parameters. Section 7 discusses the scientific background of our work, and finally Section 8 summarizes the paper contribution, draws conclusions, and outlines possible future developments of the current research activity. The main notation used throughout the paper is summarized in Table 1 to serve as a reference for the reader, with pointers to the sections where the corresponding terms are defined for the first time. A symbol has been included in Table 1 only if it is not local to the section it contains its definition, i.e., if it is referred to outside the section that contains its definition. Throughout the paper we use the operator notation $|.|$ to denote the size of both sets and tuples.

2 System model

In this section we describe the DSS characteristics we consider; in particular, we illustrate the coding operations and the attack model of malicious SNs.

2.1 Erasure codes for distributed allocation

In the scope of this paper we consider a scenario where a data unit $\mathbf{d}$ is divided into k fragments $\mathbf{d} = (d_1, \ldots, d_k)$ of equal size of z bits, then encoded onto $n \geq k$ *coded fragments* $\mathbf{y} = (y_1, \ldots, y_n)$ by a linear erasure code $\mathcal{C}(k, n)$ and such coded fragments are allocated to a distributed set of storage nodes. Any linear block code $\mathcal{C}(k, n)$ can be represented as linear mapping $\mathbf{y} = \mathbf{dG}$, through the $k \times n$ generator matrix, with addition and multiplication defined in a Galois field of a given size. In turn, the decoding process can be cast as computing k unknown d_i out of a subset of the n coded fragments. For instance, this goal can be achieved by using Gaussian elimination [18] to solve the linear problem $\mathbf{y}_\mathcal{S} = \mathbf{dG}_\mathcal{S}$, where $\mathbf{y}_\mathcal{S}$ represents a subset of the coded fragments indexed by set $\mathcal{S} \subseteq \{1, \ldots, n\}$, and $\mathbf{G}_\mathcal{S}$ are the corresponding equations, i.e. columns in the generator matrix.

Each $\mathcal{C}(k, n)$ can be characterized by the erasure code decoding probability $\epsilon(q)$ that denotes the probability that a data unit is decoded when $|\mathcal{S}| = q$,

Section 2 - Erasure code	
k	number of data unit fragments
n	number of coded fragments for a data unit
z	fragment size in bits
$\gamma(q)$	decoding probability at the q^{th} fragment
$\epsilon(q)$	decoding probability with q coded fragments
Section 2 - Storage system and allocation	
N_S	overall number of ASNs
N_M	overall number of malicious ASNs
N	number of ASNs storing n coded fragments
$\underline{n} = (n_1, \ldots, n_N)$	generic feasible allocation
A_i	i^{th} ASN in $\underline{n}$
n_i	number of coded fragments stored by A_i
$\underline{m} = (m_1, \ldots, m_N)$	generic feasible attack scenario
m_i	number of polluted coded fragments for A_i
$\mathcal{M}(\underline{m})$	malicious set for $\underline{m}$
$\mathcal{H}(\underline{m})$	honest set for $\underline{m}$
Section 3 - Virtual storage nodes	
x	size of a VSN
V_i^j	j^{th} VSN for ASN A_i
$\mathcal{N}_v(x)$	the set of all VSNs representing a data unit
Section 3 - MIND algorithm	
$\mathcal{W}(x)$	working set
w	size of $\mathcal{W}(x)$
N_{att}	max number of attempts
Section 4 - Performance indexes	
$p_i^j(\underline{n}, \underline{m})$	probability that j VSNs in A_i are polluted
$p^j(\underline{n}, \underline{m})$	probability that j VSNs are polluted
$p_{hit}^*(\underline{n}, \underline{m})$	hit probability
$a^*(\underline{n}, \underline{m})$	complexity

Table 1: Key notation used in the paper.

i.e. the probability that $\text{rank}(\mathbf{G}_{\mathcal{S}}) = k$, when $|\mathcal{S}| = q$. It shall be noted that $\epsilon(q)$ can be interpreted as the cumulative probability of decoding from q or less coded fragments. In presence of ideal Maximum Distance Separable (MDS) code, successful decoding can be accomplished from any random subset of k coded fragments and therefore $\epsilon(q) = 0$ if $q < k$ and $\epsilon(q) = 1$ if $q \geq k$.

For the broad class of RLNC in a Galois field of a given size, the analytical expressions for $\epsilon(q)$ have been derived in [17, 19]. Numerical solutions for $\epsilon(q)$ in the case of Luby transform (LT) codes is provided by [20]. In other cases, e.g. band codes [21], one may resort to simulation to estimate $\epsilon(q)$ empirically.

In the following, we will also need the probability that decoding occurs exactly when the q^{th} coded fragment is collected; this latter can be defined as $\gamma(q) = \epsilon(q) - \epsilon(q-1)$.

2.2 Pollution model

We assume that malicious storage nodes can deliberately modify the coded fragments they hold; as a consequence, the decoding results cannot be trusted in general. Even a single polluted fragment can cause significant error propagation due to the progressive combination of different received fragments during the Gaussian elimination process. Fortunately, by exploiting coding redundancy one can progressively check the decoded fragments for consistency with respect

to the redundant coded fragments; in particular, given a redundant set of coded fragments that permits decoding, i.e. a full rank $\mathbf{G}_{\mathcal{S}}$ with $|\mathcal{S}| > k$, one can exploit redundancy to detect pollution. Namely, the linear dependent equations can be checked for consistency with respect to the obtained solution. In presence of coded fragments that contradict one another the whole data block $\mathbf{d}$ is detected as polluted, even if it is not possible to identify the single polluted coded fragments. The detail of a possible implementation of the pollution detection mechanism is presented in Section 3.2.

Not all full rank $\mathbf{G}_{\mathcal{S}}$ are equally effective in the context of pollution detection and identification.

Definition 2.1. A set of coded fragments $\mathcal{S}$ is defined as *certain* if $\text{rank}(\mathbf{G}_{\mathcal{S}}) = k \wedge \forall s \in \mathcal{S} : \text{rank}(\mathbf{G}_{\mathcal{S}\setminus s}) = k$

The above definition characterizes a set of coded fragments $\mathcal{S}$ that induces a full rank generator matrix, thus allowing to decode, where there is not a single fragment $s \in \mathcal{S}$ that is essential for decoding. This amounts at never trusting one single piece of information that is potentially polluted. It turns out that when $\mathcal{S}$ is not certain one can not conclude anything on the reliability of the decoded data.

2.3 Distributed storage system

In the following we will consider a coding-based DSS that is composed of:

- N_S *actual storage nodes* (ASN) storing the n coded fragments $\mathbf{y}$ representing a data unit. A subset of N_M ASNs are *malicious* and can intentionally alter coded fragments they store.
- An arbitrary number of *clients* that can perform read or write (create) operations.
- A *master server* that manages allocations of coded fragments representing data units to ASNs and the association between names of data units and addresses of ASNs storing their coded fragments.

Clients can perform:

- a write (or create) operation. In this case, each client relies on its own data *encoder* module whose task is to create n coded fragments representing the data unit to be stored in the DSS. Subsequently, the client provides the n coded fragments to the master server. The master server comprises a data *allocator* module that randomly selects $1 \leq N \leq N_S$ ASNs to hold the n coded fragments representing the data unit; we denote as A_i $(1 \leq i \leq N)$ the i^{th} selected ASN.

  The master server completes the operation by actually transmitting the n coded fragments to the chosen ASNs according to some feasible allocation as defined in Definition 2.2 whose optimality can be defined as discussed in Section 6.3.

- A read operation. In this case, a client retrieves the addresses of ASN storing the n coded fragments of the requested data unit from the master server. Each client instructs its own data *collector* module to contact them to gather the coded fragments. Once the n coded fragments are received the client runs its data *verifier* module that attempts to identify malicious ASNs by using the algorithm presented in Section 3.2. The data verifier module of a client may either alert the master server to realize the actual removal of malicious ASNs from the DSS or it may provide the decoded clean data unit to a higher level application run by the client.

Please note that the system model we consider is important to get a global picture of how MIND can be used by clients to verify integrity of a data unit. Nevertheless, it is important to stress that for MIND to work properly the mechanism to obtain the n coded fragments is irrelevant. For instance, in a peer-to-peer based DSS the localization of ASNs storing the n coded fragments might be obtained through DHTs instead of a specialized master server. Therefore, MIND is applicable in a completely decentralized settings with no changes possibly except for the actions that should be undertaken to remove malicious ASNs from the DSS.

Let us now establish some notation that we will need in the following.

Definition 2.2. An N-tuple of positive integers $\underline{n} = (n_1, \ldots, n_N)$ is called a *feasible allocation* of n coded fragments to N ASNs if $n = \sum_{i=1}^{N} n_i$. We also define the *heterogeneity* of the feasible allocation $\underline{n}$ as its standard deviation $\sqrt{\frac{\sum_{i=1}^{N}(n_i - \frac{n}{N})^2}{N}}$. Heterogeneity quantifies how far $\underline{n}$ is from the homogeneous allocation where each ASN receives the same amount of coded fragments $\frac{n}{N}$.

Let us denote as m_i the number of polluted coded fragments stored by ASN A_i.

Definition 2.3. An N-tuple of non−negative integers $\underline{m} = (m_1, \ldots, m_N)$ is called a *feasible attack scenario* for $\underline{n}$ if $\forall i, m_i = 0$ if A_i is clean while $1 \leq m_i \leq n_i$ if A_i is malicious. We call $\mathcal{M}(\underline{m}) = \{i : A_i \text{ is malicious}\}$ the *malicious set* for $\underline{m}$ and $\mathcal{H}(\underline{m}) = \{i : A_i \text{ is honest}\}$ the *honest set* for $\underline{m}$.

3 MIND identification algorithm

In this section we describe the algorithm MIND that is run by the data verifier module of each client after the data collector has completed its task. The goal is to compute the malicious set $\mathcal{M}$ taking as input an allocation $\underline{n}$ such that pollution has been detected. MIND exploits the concept of *virtual storage node* that is a logical partition of coded fragments physically allocated to ASNs. The algorithm will return an empty malicious set $\mathcal{M}$ to indicate its failure. We also formally prove that MIND termination is guaranteed and that when MIND returns a solution it is correct with probability that can be made arbitrarily close to 1 by tuning coding parameters.

Algorithm 1 MIND $(\underline{n}, x, N_{att}, w)$

for $i = 1$ to $|\underline{n}|$ **do**
  for $j = 1$ to $\frac{n_i}{x}$ **do**
    Create VSN V_i^j
  end for
end for
for $attempt = 1$ **to** N_{att} **do**
  $\mathcal{V}(x) = \{V_i^j : 1 \leq i \leq |\underline{n}|, 1 \leq j \leq \frac{n_i}{x}\}$
  $\mathcal{W}(x) \subseteq_w^R \mathcal{V}(x)$; $\mathcal{M} = \emptyset$;
  if $(\text{rank}(\mathbf{G}_{\mathcal{W}(x)}) == k) \wedge (\neg\ \text{polluted}(\mathcal{W}(x)))$ **then**
    $\mathcal{H} = \mathcal{W}(x)$; $\mathcal{V}(x) = \mathcal{V}(x) \setminus \mathcal{W}(x)$
    for $u \in \mathcal{V}(x)$ **do**
      if $\text{polluted}(\mathcal{W}(x) \cup \{u\})$ **then**
        $\mathcal{M} = \mathcal{M} \cup \{u\}$
      else
        $\mathcal{H} = \mathcal{H} \cup \{u\}$
      end if
    end for{end of l_{in}}
    if $\text{certain}(\mathcal{H}) \wedge (\bigwedge_{\substack{\forall h \in \mathcal{H} \\ \forall m \in \mathcal{M}}} (\text{polluted}((\mathcal{H} \setminus \{h\}) \cup \{m\})))$ **then**
      break {hit: exiting and return not empty $\mathcal{M}$}
    end if
  end if
end for{end of l_{out}}
return $\mathcal{M}$

3.1 Virtual storage nodes

MIND is built around the simple idea to progressively isolate sets of coded fragments that do not trigger the pollution detection mechanism. In principle, we need to identify malicious ASN but we will show that may not always be feasible, depending on the number of fragments held by one ASN. The identification becomes possible if one partitions the set of coded fragments received by one ASN in subsets that we termed *virtual storage node* (VSN). It is worth pointing out that VSNs do not have a functional role for the DSS but are logical units introduced by MIND to detect small set of malicious fragments. In particular, given a feasible allocation $\underline{n}$, MIND partitions the set of coded fragments of each ASN A_i in VSN subsets whose cardinality is equal to a positive integer x, where $\frac{n_i}{x}$ is the number of VSNs for ASN A_i [1]. We denote as V_i^j the j^{th} VSN for ASN A_i and as $\mathcal{N}_v(x) = \{V_i^j\}$ the set of all VSNs representing a data unit. We term VSN j as *polluted* if at least one of its coded fragments is polluted.

3.2 MIND description

[1] To avoid cluttering the notation and without loss of generality in the following we assume that each value n_i in $\underline{n}$ is an integer multiple of x.

The objective of MIND is to identify a sets of coded fragments that do not trigger pollution detection. If this happen, the ASN (or VSN) contributing the selected coded fragments can be considered as honest. Moreover, once a clean set is found, it can be used as a pollution checker for all the remaining fragments: indeed if adding more fragments from one ASN (or VSN) triggers pollution detection, such ASN can be considered as malicious; in the opposite case if more fragments do not produce pollution in the decoder they can be considered as clean. The definition of VSN comprising x fragments allows us to explore different trade-offs between identification probability and computational cost.

MIND is described using pseudo-code notation in Algorithm 1: it takes as input the allocation $\underline{n}$ and VSN size x and returns an estimate of the malicious set $\mathcal{M}$. The algorithm also depends on two other parameters w and N_{att} that are introduced in the following. Algorithm 1 starts by partitioning ASN into VSN of size x (lines 1-5). Then, the core of MIND is represented by the outer loop (that we refer to in the following as l_{out}) covering lines 6-22, where one randomly looks for a clean set permitting the identification of polluters with the idea presented above. A maximum number of attempts N_{att} is established to guarantee termination. In line 7 the set of unknown VSNs $\mathcal{V}(x)$ is initialized with all the nodes. Then, a subset $\mathcal{W}(x)$ composed of w VSNs is picked randomly. From now on we term $\mathcal{W}(x)$ working set; it must be noted that $wx \geq k$ has to be considered as a condition for decoding. In line 8 we use the notation $\mathcal{W}(x) \subseteq_w^R \mathcal{V}(x)$ to represent that set $\mathcal{W}(x)$ is a random subset of $\mathcal{V}(x)$ whose cardinality is equal to w. The logical condition at line 9 serves to check if the VSNs in $\mathcal{W}(x)$ permit decoding without triggering pollution; if this is the case, set $\mathcal{W}(x)$ is considered as a set of honest $\mathcal{H}$ (line 10). In the inner loop, termed in the following l_{in} (lines 11-17), all remaining unknown nodes are checked for pollution against the clean set and moved either to the set of malicious $\mathcal{M}$ or honest $\mathcal{H}$. Finally, two sanity checks on the estimated sets $\mathcal{H}, \mathcal{M}$ are performed (logical condition in line 18): the first amounts at guaranteeing that $\mathcal{H}$ is certain according to Definition 2.1; the second one verifies the correctness of the estimated honest and polluter sets by swapping every possible honest $h \in \mathcal{H}$ with a polluter $m \in \mathcal{M}$ and checking that pollution is eventually detected.

In Algorithm 1 `certain()` and `polluted()` are boolean functions that return `true` if the property is verified by their actual parameters and `false` otherwise. The former implements Definition 2.1 while the latter is described in Algorithm 2. It relies on any decoding algorithm for the chosen encoding scheme in the distributed storage system, where function `decode`() implements the desired decoding algorithm. Then, every fragment in the set $s \in S$ is checked for consistency with the underlying linear system of equations: to this end function `substitute`() described in Algorithm 3 checks whether the obtained *solution* can be used to correctly compute the coded fragment s. In Algorithm 3 notation $g(s)_i$ is used to represent the i^{th} element of the equation to compute coded fragment s.

If s cannot be re-coded from *solution* function returns `false` and an inconsistency in the tested set S is unveiled, i.e. pollution is detected in S.

Algorithm 2 `polluted`(S)

```
1: is_polluted = false;
2: solution = decode(S);
3: for s ∈ S ∧ ¬is_polluted do
4:    if substitute(s, solution) == false then
5:       is_polluted = true;
6:    end if
7: end for
8: return is_polluted;
```

Algorithm 3 `substitute`(s,solution)

```
1: is_recoded = true;
2: f = 0;
3: for i = 1 to k do
4:    f = f + solution_i * g(s)_i
5: end for
6: if f ≠ s then
7:    is_recoded = false;
8: end if
9: return is_recoded;
```

It should be noted that function `polluted()` cannot return `true` when its argument (a set of coded fragments) does not contain polluted elements since no inconsistency is possible in this case. Conversely, function `polluted()` might fail, i.e., it could erroneously return `false`, when at least one polluted element is contained in its argument and no redundant coded fragment is available to check for inconsistencies.

3.3 MIND termination

Besides the initial double loop in lines 1-5 where creation of VSNs is performed, the core of MIND is composed of two nested loops: the inner loop l_{in} (lines 11-17) embedded in the outer loop l_{out} (lines 6-22). Both l_{in} and l_{out} are run a finite number of times proving that MIND termination is always guaranteed. Nevertheless, MIND might abort all N_{att} attempts; in that case the algorithm terminates by returning an empty malicious set $\mathcal{M}$ to indicate failure.

3.4 MIND correctness

To start proving MIND correctness we first consider the following loop invariant for l_{in}:

$$(\neg\texttt{polluted}(\mathcal{H})) \wedge (\bigwedge_{\forall m \in \mathcal{M}} \texttt{polluted}(\mathcal{H} \cup \{m\})).$$

This logical condition among l_{in} variables states that we aim at computing two not empty disjoint sets of VSNs such that pollution is not detected in set $\mathcal{H}$ and

each VSNs in $\mathcal{M}$ is able to trigger a pollution detection when joined to those in $\mathcal{H}$. We observe that:

- loop l_{in} is entered only if logical condition in line 9 is `true`, i.e., if `polluted`$(\mathcal{W}(x))$ is `false` and decoding is possible. This proves that the loop invariant is true before entering l_{in} because $\mathcal{H} = \mathcal{W}(x)$ and $\mathcal{M} = \emptyset$ in line 10.
- The loop invariant still holds true after each iteration in l_{in} because VSN $u \in \mathcal{V}(x)$ is inserted in either $\mathcal{H}$ or $\mathcal{M}$ depending on the value returned by function `polluted`$(\mathcal{W}(x) \cup \{u\})$.
- Since l_{in} is repeated a finite number of times, i.e., the cardinality of finite set $\mathcal{V}(x)$, we proved that the loop invariant is true when exiting l_{in}.

Nevertheless, the above reasoning is not sufficient to prove MIND correctness. Indeed, if we denote as $\mathcal{M}^*$ the set of actual malicious VSNs that should be returned by MIND the final step is to prove that at the end of loop l_{in} the equality $\mathcal{M} = \mathcal{M}^*$ holds. To this end we prove the following:

Lemma 3.1. *Given a set of coded fragments $\mathcal{S}$ such that* $\text{rank}(\mathbf{G}_{\mathcal{S}}) = k$ *wherein at least one element is polluted it holds that* `certain`$(\mathcal{S})$ = `true` $\rightarrow$ `polluted`$(\mathcal{S})$ = `true` *with overwhelming*[2] *probability.*

Proof. If $\mathcal{S}$ is certain then by definition $|\mathcal{S}| > k$. This also implies that for $\mathcal{S}$ to be certain it must occur that the solutions of $|\mathcal{S}|$ systems of linear equations must be obtained by excluding one coded fragment at a time in $\mathcal{S}$. For each of these solutions, the excluded coded fragment can be considered as a randomly formed one whose probability to be bitwise equal to what would be computable with the full knowledge of the solution is equal to $\frac{1}{2^z}$. The overall probability that all $|\mathcal{S}|$ solutions are successfully checked by the excluded coded fragments is then given by $\frac{1}{2^{z|\mathcal{S}|}}$; since $|\mathcal{S}| > k$ we have that $\frac{1}{2^{z|\mathcal{S}|}} < \frac{1}{2^{zk}}$. It follows that if $\mathcal{S}$ is certain and it contains at least one polluted coded fragment then the probability `polluted(`$\mathcal{S}$`)=true` is equal to $1 - \frac{1}{2^{z|\mathcal{S}|}} > 1 - \frac{1}{2^{zk}}$. Probability $1 - \frac{1}{2^{zk}}$ is thus a lower bound to the probability that at least one element in $\mathcal{S}$ triggers the detection of an inconsistency; it can be made arbitrarily close to 1 by properly choosing the coding parameters k and z. □

Furthermore, elementary properties of logical implications yield the following:

Lemma 3.2. *Given a set of coded fragments $\mathcal{S}$ such that* $\text{rank}(\mathbf{G}_{\mathcal{S}}) = k$ *wherein at least one element is polluted it holds that* `polluted`$(\mathcal{S})$ = `false` $\rightarrow$ `certain`$(\mathcal{S})$ = `false` *with overwhelming probability.*

MIND correctness can now be formally stated by analyzing the characteristics of a randomly guessed working set $\mathcal{W}(x)$ computed in line 8 such that logical condition in line 9 holds.

[2] By overwhelming probability we mean a probability that can be made exponentially close to 1.

Theorem 3.1. *Let $\mathcal{W}(x)$ be a set of coded fragments that satisfies logical condition in line 9 yielding a solution that we denote as* $\mathbf{d}$*. If all coded fragments in $\mathcal{W}(x)$ are clean then at the end of loop l_{in} we have $\mathcal{M} = \mathcal{M}^*$ with overwhelming probability.*

Proof. when a VSN $u \in \mathcal{V}(x)$ is analyzed in line 12 there are only two possibilities:

- u is composed of clean coded fragments. In this case, logical condition in line 12 `polluted(`$\mathcal{W}(x) \cup \{u\}$`)=false` since coded fragments in u surely satisfy $\mathbf{d}$. Therefore, insertion of u in $\mathcal{H}$ is correctly performed.
- u contains $1 \leq x_p \leq x$ polluted coded fragments therefore $u \in \mathcal{M}^*$. As we did in the proof of Lemma 3.1, we consider the x_p polluted coded fragments in u as randomly formed with respect to $\mathbf{d}$; this implies that the probability `polluted(`$\mathcal{W}(x) \cup \{u\}$`)=true` in line 12 is equal to $1 - \frac{1}{2^{z x_p}}$. In this case, MIND correctly inserts u in $\mathcal{M}$. In the unlikely case all x_p polluted coded fragments satisfy $\mathbf{d}$ (the probability of this event is equal to $\frac{1}{2^{z x_p}}$) we have that $u \notin \mathcal{M}^*$ therefore it is correct to consider u as a member of set $\mathcal{H}$, instead.

The above analysis suggests that classification of VSNs in $\mathcal{V}(x)$ is correctly performed with overwhelming probability and we obtain $\mathcal{M} = \mathcal{M}^*$ at the end of loop l_{in}. □

We complete the analysis of MIND correctness by proving the following theorem that shows that a wrong guess for $\mathcal{W}(x)$ computed in line 8 such that logical condition in line 9 holds will lead to an aborted attempt with overwhelming probability.

Theorem 3.2. *Let $\mathcal{W}(x)$ be a set of coded fragments that satisfies logical condition in line 9. If $\mathcal{W}(x)$ contains at least one polluted coded fragment then set $\mathcal{H}$ at the end of loop l_{in} is not certain with overwhelming probability.*

Proof. The set of coded fragments $\mathcal{W}(x)$ satisfies all hypothesis stated to derive Lemma 3.2 therefore we conclude it is not certain with overwhelming probability. Furthermore, at the end of loop l_{in} we have $\mathcal{W}(x) \subseteq \mathcal{H}$ with h VSNs that have been added to $\mathcal{H}$:

- if $h = 0$ then $\mathcal{H} = \mathcal{W}(x)$ hence by Lemma 3.2 $\mathcal{H}$ is not certain with overwhelming probability.
- If $h > 0$ then $\mathcal{H} \supset \mathcal{W}(x)$. A VSN u is added to $\mathcal{H}$ only if `polluted`$(\mathcal{W}(x) \cup \{u\}) =$ `false` therefore at the end of loop l_{in} we have `polluted`$(\mathcal{H}) =$ `false`. It follows that $\mathcal{H}$ verifies the hypothesis of Lemma 3.2, i.e., it is full rank and it contains at least one polluted coded fragment, therefore it is not certain with overwhelming probability.

This proves that $\mathcal{H}$ is not certain with overwhelming probability every time a wrong guess is made in line 8; in this case MIND aborts the attempt when evaluating logical condition in line 18. □

To conclude, our theorems show that when MIND returns a solution $\mathcal{M}$ it is correct with overwhelming probability which can be made arbitrarily close to 1 by tuning coding parameters k and z.

Remark:

Although MIND is an extremely simple algorithm it includes a tricky part, i.e., the verification that the set of (hopefully) clean equations $\mathcal{H}$ is actually pollution free at the end of loop l_{in}. There is no guarantee this property is always verified: indeed, it is possible to compute $\mathcal{H}$ such that it contains polluted equations.

As a simple example consider a data unit $\mathbf{d} = (d_1, \ldots, d_k)$ and the data encoder creating $n \geq k$ coded fragments $\mathbf{y} = (y_1, \ldots, y_n)$. Further assume that MIND is run at the equation level, i.e., when the size of VSN is $x = 1$. Due to the randomness inherent in both the encoding process and in the choice of elements to be possibly included in $\mathcal{H}$ (line 8 of Algorithm 1) it might happen that at the $k-1^{th}$ iteration of l_{in} a subset of $k-1$ linearly independent coded fragments allows one to recover $k-1$ out of k data fragments (we denote as $d_{unknown}$ the data fragment yet to be recovered). At the k^{th} iteration there is a non zero probability that a degree 1 coded fragment $y_{k^{th}} = d_{unknown}$ held by a malicious ASN is selected. In this case, a corrupted data unit is recovered (the malicious ASN has turned $y_{k^{th}} = d_{unknown}$ into $y_{k^{th}} = d_{corrupted}$) and all subsequent coded fragments would be considered by MIND as polluted since full and blind trust would be given to the correctness of the process leading to set $\mathcal{H}$.

The problem is: how can we discriminate between such a case and the case where coded fragment $y_{k^{th}} = d_{unknown}$ is held by an honest ASN? The answer is: we cannot, unless we define some computable property of set $\mathcal{H}$ that allows one to discard or accept the solution provided at the end of inner loop l_{in}. The solution we propose is to define the *certain* property (Definition 2.1) that can be efficiently verified on set $\mathcal{H}$ and to formally prove that we can correctly use it to discard or accept a solution for $\mathcal{H}$ (lemmas and theorems in Section 3.4).

4 MIND performance

In this section we develop a mathematical characterization of the performance of a size N, feasible allocation $\underline{n}$ in a coding-based DSS that is composed of N_M malicious ASNs and $N_S - N_M$ honest ASNs. Performance is defined over two indexes: the *hit probability* (the fraction of times MIND is able to successfully isolate malicious VSNs), and *complexity* (the average number of trials before MIND succeeds).

4.1 Hit probability and complexity with respect to a feasible attack scenario

In this section we first focus on a particular feasible attack scenario $\underline{m}$ for $\underline{n}$ with the goal to analytically derive the hit probability and complexity of MIND run on $\underline{n}$ and $\underline{m}$ as functions of both the VSN size x and the maximum number of attempts N_{att}. For the sake of readability and to avoid cluttering the notation we omit explicit dependencies of all derived formulas on both x and N_{att}. Furthermore, without loss of generality in the following we assume that each value n_i in $\underline{n}$ is an integer multiple of x.

4.1.1 Probability distribution of the number of polluted VSNs

For a given $\underline{n}$ we first focus on ASN A_i to derive the probability distribution $p_i^j(\underline{n},\underline{m})$, i.e., the probability that j VSNs for ASN A_i are polluted.

We denote as $\underline{l_i} = (l_1, \ldots, l_{\frac{n_i}{x}})$ an $\frac{n_i}{x}$-tuple of non negative integers to represent a possible distribution of polluted coded fragments among VSNs of ASN A_i where $m_i = \sum_{h=1}^{\frac{n_i}{x}} l_h$. We also denote as $\mathcal{L}_i$ the set of all such distributions of polluted coded fragments for ASN A_i.

For a given distribution of polluted coded fragments $\underline{l_i}$ we call $J(\underline{l_i}) = \sum_{h=1}^{\frac{n_i}{x}} \mathbf{1}\{l_h > 0\}$ the number of polluted VSNs for distribution $\underline{l_i}$ where the symbol $\mathbf{1}\{A\}$ represents the indicator function whose value is equal to 1 if statement A is true and 0 otherwise. Given $\underline{l_i}$ there exist $\prod_{h=1}^{\frac{n_i}{x}} \binom{n_i^h}{l_h}$ possible ways of spreading polluted coded fragments. It follows that $p_i^j(\underline{n},\underline{m})$ can be expressed as

$$p_i^j(\underline{n},\underline{m}) = \frac{\sum_{\underline{l_i}\in\mathcal{L}_i} \left[\prod_{h=1}^{\frac{n_i}{x}} \binom{n_i^h}{l_h}\right] \mathbf{1}\left\{J(\underline{l_i}) = j\right\}}{\binom{n_i}{m_i}}. \tag{1}$$

Please note that if ASN A_i is honest then $p_i^j(\underline{n},\underline{m}) = 1$ for $j = 0$ and $p_i^j(\underline{n},\underline{m}) = 0$ otherwise.

Starting from $p_i^j(\underline{n},\underline{m})$ it is possible to derive the probability distribution $p^j(\underline{n},\underline{m})$, i.e., the probability that j VSNs are polluted in a feasible allocation $\underline{n}$ under a feasible attack scenario $\underline{m}$ for $\underline{n}$, as

$$p^j(\underline{n},\underline{m}) = \sum_{j_{i_1},\ldots,j_{i_{|\mathcal{M}(\underline{m})|}}} \left[\prod_{i\in\mathcal{M}(\underline{m})} p_i^{j_i}(\underline{n},\underline{m})\right] \mathbf{1}\left\{\sum_{i\in\mathcal{M}(\underline{m})} j_i = j\right\}. \tag{2}$$

It follows that the average number of polluted VSNs is given by

$$\overline{j}(\underline{n},\underline{m}) = \sum_{j=1}^{|\mathcal{N}_v(x)|} j p^j(\underline{n},\underline{m}),$$

and the average number of coded fragments to discard after identification of polluted VSNs is

$$\overline{d}(\underline{n},\underline{m}) = x \sum_{j=1}^{|\mathcal{N}_v(x)|} j p^j(\underline{n},\underline{m}).$$

4.1.2 Upper bound on the hit probability

When j VSNs are polluted only a subset $\mathcal{R}(x,j) \subseteq \mathcal{N}_v(x)$ of the initial set of VSNs (where $|\mathcal{R}(x,j)| = |\mathcal{N}_v(x)| - j$) remains to recover the original data unit.

An upper bound on the hit probability depends on the probability to decode the original clean data unit by using the set of remaining clean VSNs $\mathcal{R}(x,j)$ that must also be certain according to Definition 2.1 (this condition is checked by the left conjunct in line 18 of Algorithm 1). As discussed in Section 2.1, the probability to decode the original clean data unit in this case is given by $\epsilon(x(|\mathcal{N}_v(x)| - j))$, i.e., the decoding probability with the $x(|\mathcal{N}_v(x)| - j)$ coded fragments contained in $\mathcal{R}(x,j)$. The probability that the set of coded fragments $\mathcal{R}(x,j)$ is certain is equal to $\left(\frac{\epsilon(x(|\mathcal{N}_v(x)|-j)-1)}{\epsilon(x(|\mathcal{N}_v(x)|-j))}\right)^{x(|\mathcal{N}_v(x)|-j)}$, i.e., the conditional probability that all possible subsets of coded fragments in $\mathcal{R}(x,j)$ of cardinality equal to $x(|\mathcal{N}_v(x)| - j) - 1$ allow one to recover the original data, given that the set of coded fragments in $\mathcal{R}(x,j)$ is decodable.

It follows that when j VSNs are polluted recovery of the original data unit is feasible and certain, in the sense of Definition 2.1, with probability

$$p_{certain}(j) = \epsilon(x(|\mathcal{N}_v(x)| - j)) \left(\frac{\epsilon(x(|\mathcal{N}_v(x)| - j) - 1)}{\epsilon(x(|\mathcal{N}_v(x)| - j))}\right)^{x(|\mathcal{N}_v(x)|-j)} \tag{3}$$

As discussed in Section 3, MIND is based on the concept of the working set whose elements are VSNs and whose size is equal to w.[3] When j VSNs are polluters and subset $\mathcal{R}(x,j)$ is certain (as defined in (2.1)) then MIND can be successful only if at least one subset $\mathcal{W}(x) \subseteq \mathcal{R}(x,j)$ (where $|\mathcal{W}(x)| = w$) is decodable.

The conditional probability that a subset $\mathcal{W}(x) \subseteq \mathcal{R}(x,j)$ is decodable given that $\mathcal{R}(x,j)$ is certain can be expressed as

$$p_{decode}(j,w) = \begin{cases} \frac{\epsilon(wx)}{\left(\frac{\epsilon((|\mathcal{N}_v(x)|-j)x-1)}{\epsilon((|\mathcal{N}_v(x)|-j)x)}\right)^{(|\mathcal{N}_v(x)|-j-w)x}} & \text{if } w < |\mathcal{N}_v(x)| - j \\ 1 & \text{if } w = |\mathcal{N}_v(x)| - j \end{cases}. \tag{4}$$

Hence, the probability that at least one subset $\mathcal{W}(x) \subseteq \mathcal{R}(x,j)$ is decodable is given by

$$p_{exist}(j,w) = 1 - (1 - p_{decode}(j,w))^{\binom{|\mathcal{N}_v(x)|-j}{w}}. \tag{5}$$

It follows that the product of probabilities (3) and (5) represents an upper bound on the hit probability of MIND when j VSNs are polluted obtained as

$$p^{ub}_{hit}(j,w) = p_{certain}(j) p_{exist}(j,w). \tag{6}$$

[3] Again, w actually depends on x but we omit it to improve readability.

4.1.3 An expression for the hit probability

If j VSNs are polluted and at least one subset $\mathcal{W}(x) \subseteq \mathcal{R}(x,j)$ whose size is equal to w is decodable, then the probability that a clean and decodable working set is selected to start computing solutions $\mathcal{H}$ and $\mathcal{M}$ (logical condition in line 9) is given by

$$p_{select}(j,w) = \frac{\binom{|\mathcal{N}_v(x)|-j}{w}}{\binom{|\mathcal{N}_v(x)|}{w}} \frac{p_{decode}(j,w)}{p_{exist}(j,w)}, \tag{7}$$

where the left factor $\frac{\binom{|\mathcal{N}_v(x)|-j}{w}}{\binom{|\mathcal{N}_v(x)|}{w}}$ represents the probability of randomly selecting a working set composed of clean VSNs while the right factor $\frac{p_{decode}(j,w)}{p_{exist}(j,w)}$ is the conditional probability such clean working set is decodable given that at least one exists. It follows that the probability MIND will succeed in N_{att} attempts is given by

$$p_{success}(j,w) = \sum_{i=1}^{N_{att}} p_{select}(j,w)(1-p_{select}(j,w))^{i-1}, \tag{8}$$

i.e., the sum of probabilities a truncated geometric random variable is equal to i.

All previous derivations lead us to write that MIND is successful when j VSNs are polluted with probability

$$p_{hit}(j,w) = p_{hit}^{ub}(j,w) p_{success}(j,w). \tag{9}$$

Then, we can compute the *average hit probability* of MIND as

$$\overline{p}_{hit}(\underline{n},\underline{m},w) = \sum_{j=1}^{|\mathcal{N}_v(x)|} p_{hit}(j,w) p^j(\underline{n},\underline{m}). \tag{10}$$

If we denote the value of w that maximizes (10) as w^*, i.e., $w^* = \arg\max_w \overline{p}_{hit}(\underline{n},\underline{m},w)$, then we refer to

$$p_{hit}^*(\underline{n},\underline{m}) = \overline{p}_{hit}(\underline{n},\underline{m},w^*) \tag{11}$$

as the *optimal hit probability* of MIND for feasible allocation $\underline{n}$ under feasible attack $\underline{m}$.

4.1.4 An expression for the average complexity

When MIND is successful it terminates at the i^{th} attempt with probability $\frac{p_{select}(j,w)(1-p_{select}(j,w))^{i-1}}{p_{success}(j,w)}$, therefore the average number of attempts before hitting the successful one is equal to

$$a(j,w) = \frac{\sum_{i=1}^{N_{att}} i p_{select}(j,w)(1-p_{select}(j,w))^{i-1}}{p_{success}(j,w)}.$$

We can express the average complexity (average number of attempts before hit) of MIND as

$$\overline{a}(\underline{n}, \underline{m}, w) = \sum_{j=1}^{|\mathcal{N}_v(x)|} a(j, w) p^j(\underline{n}, \underline{m}). \tag{12}$$

Also in this case, we refer to

$$a^*(\underline{n}, \underline{m}) = \overline{a}(\underline{n}, \underline{m}, w^*) \tag{13}$$

as the *optimal complexity* of MIND for feasible allocation $\underline{n}$ under feasible attack $\underline{m}$.

5 Validation results

In this section we present validation results for the mathematical model we developed in Section 4. To this end, we developed a C++ prototype for MIND whose results are compared against predictions from Equations 11 and 13 to assess its accuracy.

We focus on a reference scenario where a data unit is partitioned in $k = 32$ fragments whose size in bits is $z = 32$ and where the encoder creates coded fragments employing RLNC in a Galois field of size 2, i.e. random network coding using XOR operations to linearly combine fragments. In case of RLNC we can use to the analytical results derived in [17] for $\epsilon(q)$ that is required by our model. We consider two feasible allocations $\underline{n}_1$ and $\underline{n}_2$; for each one we analyze four feasible attack scenarios. In particular, we assume:

- the data allocator randomly selects $N = 4$ ASNs and determines the feasible allocation $\underline{n}_1 = (32, 16, 8, 4)$ where $n = 60$. In this case, we assume there is a single malicious ASN, i.e., $M = 1$, that alters four coded fragments. We consider all possible feasible attack scenarios ($\underline{m}_{1,1} = (4, 0, 0, 0)$, $\underline{m}_{1,2} = (0, 4, 0, 0)$, $\underline{m}_{1,3} = (0, 0, 4, 0)$, and $\underline{m}_{1,4} = (0, 0, 0, 4)$) that are obtained by assuming one out of four ASNs in $\underline{n}_1$ is malicious.

- The data allocator randomly selects $N = 8$ ASNs and determines the feasible allocation $\underline{n}_2 = (20, 12, 8, 8, 4, 4, 4, 4)$ where $n = 64$. In this case, we assume there are two malicious ASNs, i.e., $M = 2$, that alter two coded fragments each. We choose four out of $\binom{N}{2}$ possible feasible attack scenarios ($\underline{m}_{2,1} = (2, 0, 0, 0, 2, 0, 0, 0)$, $\underline{m}_{2,2} = (0, 2, 0, 0, 2, 0, 0, 0)$, $\underline{m}_{2,3} = (0, 0, 2, 0, 2, 0, 0, 0)$, and $\underline{m}_{2,4} = (0, 0, 0, 0, 2, 2, 0, 0)$).

We ran 1,000,000 trials of the C++ prototype for each feasible attack scenario and computed the estimate for $p^*_{hit}()$ as the fraction of trials that successfully terminated, and the estimate for $a^*()$ as the average value of the number of attempts before hitting the successful one.

The illustrations in Figure 1 show that the values of $p^*_{hit}()$ predicted by Equation 11 are in excellent agreement with the results obtained by the C++

implementation of MIND for all considered scenarios (lines describe model predictions and points represent prototype performance results: these two sets of values *always overlap*). The same conclusions can be drawn from results depicted in Figure 2 showing MIND complexity as predicted by Equation 13. We conclude that our mathematical characterization of MIND optimal hit probability and complexity is highly accurate; we will exploit the model predictions in Section 6 to compute optimal feasible allocations for a coding-based DSS for several values of the system parameters.

wx	$x = 1$	$x = 2$	$x = 4$
32	0.041988	0.209195	0.466666
36	0.031790	0.151724	0.4
40	0.009935	0.103448	0.333333
44	0.003732	0.064367	0.266666
48	0.001015	0.034482	0.2
52	0.000143	0.013793	0.133333
56	0.000002	0.002298	0.066666

Table 2: Probability $p_{select}(\frac{4}{x}, w)$ in Equation 7 as function of x for $\underline{n}_1$ and $\underline{m}_{1,4}$.

Further inspection of results depicted in Figures 1 and 2 suggests a few more observations: heterogeneity plays an important role in the severity of the pollution attack. Indeed, although for both feasible allocations we considered the same number of coded fragments is altered, the success of MIND heavily depends on *which* ASNs are malicious in the feasible attack scenarios. For instance, for $\underline{n}_1$ the feasible attack scenario $\underline{m}_{1,4}$ (where the malicious ASN controls only 4 out of 60 fragments) yields higher p^*_{hit} for all values of parameter x. We can also observe that in the same case Equation 2 yields $p^{\frac{4}{x}}(\underline{n}_1, \underline{m}_{1,4}) = 1$ for all values of x we consider; therefore, we conclude that $p_{certain}(\frac{4}{x})$, $p_{decode}(\frac{4}{x}, w)$, and $p_{exist}(\frac{4}{x}, w)$ do not depend on x for a fixed value of the product wx. It follows that $p_{select}(\frac{4}{x}, w)$ exclusively depends on the ratio of binomial coefficients $\frac{\binom{|\mathcal{N}_v(x)|-\frac{4}{x}}{w}}{\binom{|\mathcal{N}_v(x)|}{w}}$; this is an increasing function of x as witnessed by results in Table 2.

In Table 3 we show $p^j(\underline{n}_1, \underline{m}_{1,*})$ for the four feasible attack scenarios we consider: it can be noted that for $\underline{m}_{1,1}$ Equation 2 exhibits a behavior that is less straightforward. In this case it is true that Equation 6 provides values that decrease as x increases (hence for $N_{att} \to \infty$ the value of Equation 11 decrease as x increases). Nonetheless, for small values of N_{att} a larger size for VSNs is beneficial as it yields higher values of Equation 11.

Some remarks

The following remarks complement these validation results:

$\underline{n}_1 = (32, 16, 8, 4)$				
j	$p^j(\underline{n}_1, \underline{m}_{1,1})$	$p^j(\underline{n}_1, \underline{m}_{1,2})$	$p^j(\underline{n}_1, \underline{m}_{1,3})$	$p^j(\underline{n}_1, \underline{m}_{1,4})$
1	0.00022	0.00219	0.02857	1
2	0.05295	0.22418	0.87143	0
3	0.44849	0.63297	0	0
4	0.49834	0.14066	0	0
$\overline{j}(\underline{n}_1, \underline{m})$	3.44494	2.91209	1.97143	1

Table 3: Probability distribution in Equation 2 for $x = 4$ in feasible allocation $\underline{n}_1$.

- for a fixed number of polluted coded fragments the average number of polluted VSNs $\overline{j}(\underline{n}, \underline{m})$ is always less than the number of polluted coded fragments as witnessed by results in Table 3. In turn, this means that the probability that a clean and decodable working set is selected to start computing solutions $\mathcal{H}$ and $\mathcal{M}$ (logical condition in line 9) increases as x increases as shown by results in Table 2. To summarize: the higher x the lower the overall number of VSNs, the lower the average number of polluted VSNs, the higher p_{select}, the higher $p_{success}$ (Equation 8), and finally the higher p_{hit}.

- Since our evaluation of MIND performance and complexity is model based we developed a non-optimized C++ implementation with the sole purpose to validate analytical results. For this reason, we did not strive to realize a highly optimized software that would be suitable for a thorough evaluation of running times. Nevertheless, we performed some tests to provide a ballpark figure of running times. Figure 3 shows the average CPU time in milliseconds for a single successful run of MIND on a PC equipped with an i9 CPU. We only show the average CPU time for a successful trial of MIND, i.e., a trial wherein logical condition in line 9 of Algorithm 1 is `true`, because the average CPU time for a failed trial takes much less than 1 millisecond (it takes only a few tens of microseconds to run instructions realizing lines 7-9). This means that the overall running time of MIND can be well approximated by the CPU time for a successful trial. Results depicted in Figure 3 refer to feasible allocations $\underline{n}_1$ and $\underline{n}_2$ and all feasible attacks $\underline{m}_{1,*}$ and $\underline{m}_{2,*}$. All parameter values are those defined in the reference scenario to compute validation results.

  In a successful trial, most of the CPU time is spent for the evaluation of logical condition when internal loop l_{in} is exited. For a fixed value of k, the CPU time to evaluate this logical condition is mostly determined by the number of VSNs since the logical AND of evaluations of function `polluted`$((\mathcal{H} \setminus \{h\}) \cup \{m\})$ must be performed by swapping every possible honest VSN $h \in \mathcal{H}$ with a polluter VSN $m \in \mathcal{M}$. Since VSNs are size x random subsets of the set of coded fragments stored by SNs the higher x

the lower the number of VSN to create (lines 1-5), test for integrity (lines 11-17), and verify for correctness of the solution (line 18). Therefore, it can be easily noted that running times decrease as x increases since the number of VSN decreases as shown in Figure 3.

Furthermore, we observe that by doubling N (the number of ASNs used by feasible allocations $\underline{n}_1$ and $\underline{n}_2$) and by doubling the number of malicious ASN (from 1 in $\underline{n}_1$ to 2 in $\underline{n}_2$) the average CPU times of MIND are only slightly increased.

- the system model we consider is such that the task of identifying malicious SNs is distributed to clients at each read attempt from the DSS. This means that if at any given time any C read operations have been attempted on any data units stored by the DSS then the probability of at least one client spotting at least one malicious SN is given by $p_{spot}(C) = 1-(1-p)^C$ where p can either be $p^*_{hit}()$ as defined in Equation 11 if we focus on a particular feasible allocation $(\underline{a}, \underline{x})$ under a feasible attack $(\underline{a}, \underline{x}, \underline{m})$, or it can be $h^{opt}(N_M, N_S, N)$, i.e., the optimal hit probability for MIND as defined in Section 6.1. Probability $p_{spot}(C)$ can be very close to 1 even for small values of p, i.e., for small values of parameter N_{att} that yields a very low time complexity. For instance, the lowest values for $p^*_{hit}()$ in Figure 1 are obtained when $N_{att} = 10$ and can be lower bounded by 0.2. In this case, after only $C = 10$ overall read attempts by clients the probability of spotting at least one malicious SNs would be equal to 0.892 while after $C = 20$ read attempts it would rise to 0.988.

- although we only show results for one reference scenario, two feasible allocations, and four feasible attack scenarios we verified that predictions of Equations 11 and 13 are extremely accurate in a broad range of values of parameters N, M, and n. Furthermore, very high accuracy has been obtained for several combinations of feasible allocations and feasible attack scenarios.

6 Analysis results

In this section we present an example of applicability of our work and we exploit the mathematical characterization of MIND performance developed in Section 4 to determine the optimal feasible allocation $\underline{n}^*$ (hence its optimal size N^*) for a DSS where N_M out of N_S ASNs are malicious. To this end, we define the concepts of performance and collusion resistance of a feasible allocation in Sections 6.1 and 6.2, respectively. In Section 6.3 we state the optimization problems to obtain optimal feasible allocations and in Section 6.4 we present and comment on results we obtain.

6.1 Overall performance of a feasible allocation

In the following section we define the performance provided by allocation $\underline{n}$ with respect to all possible attack scenarios that could be launched by N_M malicious ASNs in a DSS. Given a feasible attack scenario $\underline{m}$ for $\underline{n}$ we define its *degree* as $|\mathcal{M}(\underline{m})|$, i.e., the number of malicious ASNs in $\underline{n}$. We denote as $\Theta_{\underline{n}}^{M}$ the set of all degree M feasible attack scenarios and it is easy to show that $|\Theta_{\underline{n}}^{M}| = \sum_{\underline{m}\in\Theta_{\underline{n}}^{M}} \left(\prod_{i\in\mathcal{M}(\underline{m})} n_i \right)$.

Definition 6.1. A feasible attack scenario $\underline{m}$ for $\underline{n}$ is said to be *maximally harsh* for $\underline{n}$ if $\forall i \in \mathcal{M}(\underline{m}), m_i = n_i$. We denote as $\Phi_{\underline{n}}^{M}$ the set of all possible degree M, maximally harsh, feasible attack scenarios for $\underline{n}$. It is easy to show that $\Phi_{\underline{n}}^{M} \subseteq \Theta_{\underline{n}}^{M}$ and that $|\Phi_{\underline{n}}^{M}| = \binom{N}{M}$. Furthermore, $\forall \underline{m} \in \Phi_{\underline{n}}^{M}$ there exist $\left(\prod_{i\in\mathcal{M}(\underline{m})} n_i \right) - 1$ feasible attack scenarios $\underline{m}^*$ such that $\mathcal{M}(\underline{m}^*) = \mathcal{M}(\underline{m})$.

We consider the set $\Phi_{\underline{n}}^{M}$ of all possible degree M, maximally harsh, feasible attack scenarios for $\underline{n}$ and we define the hit probability of $\underline{n}$ as

$$h^*(\underline{n}, N, M) = \frac{1}{\binom{N}{M}} \sum_{\underline{m}\in\Phi_{\underline{n}}^{M}} p_{hit}^*(\underline{n}, \underline{m}).$$

We also define the complexity of $\underline{n}$ as

$$c^*(\underline{n}, N, M) = \frac{1}{\binom{N}{M}} \sum_{\underline{m}\in\Phi_{\underline{n}}^{M}} a^*(\underline{n}, \underline{m}).$$

Then we can define the average hit probability of allocation $\underline{n}$ when $1 \leq M \leq N_M$ selected SNs are malicious provided $\underline{n}$ includes at least one malicious SN as

$$h(\underline{n}, N_M, N_S, N) = \frac{\sum_{M=1}^{N_M} H(M, N_M, N_S - N_M, N) h^*(\underline{n}, N, M)}{1 - H(0, N_M, N_S - N_M, N)}, \qquad (14)$$

where $H(M, N_M, N_S - N_M, N)$ denotes the probability that a hypergeometric random variable is equal to M when population sizes are N_M and $N_S - N_M$, and a subset of size N is randomly selected. Analogously, its complexity is defined as

$$c(\underline{n}, N_M, N_S, N) = \frac{\sum_{M=1}^{N_M} H(M, N_M, N_S - N_M, N) c^*(\underline{n}, N, M)}{1 - H(0, N_M, N_S - N_M, N)}. \qquad (15)$$

6.2 Collusion resistance of a feasible allocation

In this section we introduce concepts allowing us to tell how likely $\underline{n}$ is going to resist a colluding attack in a storage system with N_M malicious ASNs. We define $CP(\underline{n}, \underline{m}) = \sum_{i \in \mathcal{M}(\underline{m})} n_i$ as the *colluding potential* of the feasible attack scenario $\underline{m}$ for $\underline{n}$, i.e., the overall number of polluted coded fragments representing a data unit that are controlled by the set of malicious ASNs storing a data unit. Furthermore, given a feasible allocation $\underline{n}$ and a feasible attack scenario $\underline{m}$ for $\underline{n}$

Definition 6.2. we consider $\underline{n}$ as *collusion resistant* to $\underline{m}$ if $CP(\underline{n}, \underline{m}) < k$, i.e., if all malicious ASNs cannot collude to modify the original data unit.

It is easy to show that, $\forall \underline{m} \in \Phi_{\underline{n}}^{M}$ there exist $(\prod_{i \in \mathcal{M}(\underline{m})} n_i) - 1$ feasible attack scenarios $\underline{m}^*$ such that $CP(\underline{n}, \underline{m}^*) = CP(\underline{n}, \underline{m})$ for any value of M. For this reason, we only consider maximally harsh, feasible attack scenarios for $\underline{n}$ to define that:

Definition 6.3. feasible allocation $\underline{n}$ is *M-collusion resistant* if $\forall \underline{m} \in \Phi_{\underline{n}}^{M}$, $\underline{n}$ is collusion resistant to $\underline{m}$.

The concept of $M-$collusion resistance of $\underline{n}$ can be formalized by the collusion resistance function defined as

$$r(\underline{n}, M) = \prod_{\underline{m} \in \Phi_{\underline{n}}^{M}} \mathbf{1}\left\{CP(\underline{n}, \underline{m}) < k\right\},$$

where collusion resistance $r(\underline{n}, M) = 1$ if and only if $\underline{n}$ is $M-$collusion resistant.

6.3 Optimal feasible allocation

Finally, in this section we consider the problem of determining the optimal allocation $\underline{n}^*$ over N ASNs for a DSS composed of N_S ASNs with a fixed number N_M of malicious ASNs. To this end we choose $\underline{n}^*$ that maximizes $h(\underline{n}, N_M, N_S, N)$ and that is N_M-collusion resistant, i.e.,

$$\underline{n}^*(N_M, N_S, N) = \arg\max_{\underline{n}} h(\underline{n}, N_M, N_S, N) r(\underline{n}, N_M). \tag{16}$$

We denote as

$$h^{opt}(N_M, N_S, N) = h(\underline{n}^*(N_M, N_S, N), N_M, N_S, N) \tag{17}$$

the optimal value of the hit probability and as

$$c^{opt}(N_M, N_S, N) = c(\underline{n}^*(N_M, N_S, N), N_M, N_S, N) \tag{18}$$

the optimal complexity of MIND in such a scenario.

Parameter	Values
k	32
n	64
N_S	32, 96
N_M	2, ... ,$N_S/4$
x	1, 2, 4, 8

Table 4: Parameter setting for optimization.

6.4 Results

Table 4 summarizes the values of parameters we considered in this investigation. We assume RLNC in a Galois field of size 2 for coded fragment creation.

Results we discuss in this section can also be considered as a security analysis of the DSS under a pollution attack mounted by N_M out of N_S malicious ASNs. In particular, the output of our analysis is composed of feasible allocations of coded fragments representing a data unit that not only are optimal with respect to the probability MIND is able to correctly isolate polluted VSNs but are also resistant to collusion as defined in Definition 6.3.

The first set of results is depicted in Figure 4 and compares optimal performance of MIND for the range of values of x we consider and for $N_S = 32$ and $N_{att} = 50$. It can be noted that:

- for $x = 4$ and $x = 8$ accuracy increases as the size of the optimal allocations N increases. Nevertheless, the benefit yielded by higher values of N are counterbalanced by the increased likelihood of stragglers [4], i.e., the time taken to retrieve the slowest, or straggling, coded fragments dominates the decoding of the data unit. Nonetheless, results suggest that in both cases the optimal allocation for h^{opt} is the homogeneous allocation, i.e., $N^* = \frac{n}{x}$ and $\underline{n}^* = (x, \ldots, x)$.

- For $x = 1$ and $x = 2$ the trend exhibited by h^{opt} *does* depend on N_M. Indeed, when the number of malicious ASNs in the system is low ($N_M = 2$) accuracy of MIND is monotonically increasing with N, while when N_M is large ($N_M = 8$) accuracy *decreases* as N increases, instead. For intermediate values of the number of ASNs compromised during the attack ($4 \leq N_M \leq 6$) MIND accuracy is not monotonic with N anymore.

- For $x = 2$ optimal allocations are:
  - $N^* = \frac{n}{x}$ and $\underline{n}^* = (x, \ldots, x)$, for $N_M = 2$;
  - for all other cases the optimal allocation turns out to be $\underline{n}^* = (n - (N^*-1)x, x, \ldots, x)$ where $N^* = 21$ for $N_M = 4$, $N^* = 23$ for $N_M = 6$, and $N^* = 25$ for $N_M = 8$. Please note that the above optimal allocation is also the *most heterogeneous* feasible allocation, i.e., the one with the highest value for the standard deviation of $\underline{n}^*$.

- For $x = 1$ optimal allocations are:
  - $N^* = \frac{n}{x}$ and $\underline{n}^* = (x, \ldots, x)$, for $N_M = 2$;
  - for $N_M = 4$ we obtain $N^* = 24$ and optimal allocation is $\underline{n}^* = (10, 7, 7, 7, 7, 7, 2, 1, 1, \ldots, 1)$. This is neither the most nor the least heterogeneous feasible allocation;
  - for $N_M = 6$ and $N_M = 8$ we get $N^* = 17$ and $N^* = 19$, respectively. In both cases the optimal allocation is $\underline{n}^* = (n-3(N^*-1), 3, 3, \ldots, 3)$ that is also the most heterogeneous one.

Figure 5 shows the same results as a function of x for $N_M = N_S/4$. It can be noted that, optimal allocations can be found for the highest value of N_M only for $x = 1$ and $x = 2$. Higher values of x make MIND faster and more accurate but weaken it since optimal allocations can only be $6-$resistant and $2-$resistant for $x = 4$ and $x = 8$, respectively. To summarize, it is possible to find at least one allocation that is collusion resistant up to $N_M = 8 = \frac{N_S}{4}$; this means that the DSS is able to resist to attacks launched by no more than 25% of malicious ASNs.

Finally, Figure 6 shows the same results as a function of N_{att} for $x = 2$ and for different sizes of the DSS N_S. It can be noted that MIND optimal performance does depend on the maximum number of attempts N_{att}. In particular, for low-to-moderate values of pollution in the system, i.e., $N_M \leq 6$, optimal performance of MIND increase monotonically as N increases when the algorithm can be run for a very large number of attempts, i.e., for $N_{att} = 1000$. This means that the homogeneous allocation where $N^* = \frac{n}{x}$ is the optimal feasible allocation. This is in sharp contrast with the previous observation we made on results depicted in Figure 4 where an optimal value $N^* < \frac{n}{x}$ was identified for $N_{att} = 50$ when the number of malicious ASN is $N_M > 2$. It follows that optimal allocations also depend on the time constraints imposed on the detection and identification process.

The above observations do not hold in a small size DSS ($N_S = 32$) when pollution peaks at its maximum, i.e., when $N_M = N_S/4$; in this case the optimal feasible allocation is not the homogeneous one and $N^* = 25$ and $\underline{n}^* = (n - x(N^* - 1), x, x, \ldots, x)$ represent the best solution for any time constraint on MIND maximum complexity.

Remark

In this section we simply aimed at giving an example of how to exploit the accurate characterization of MIND performance to evaluate the resilience of a DSS wherein a given number of malicious SN is assumed. The computational cost to evaluate formulas yielding MIND performance, complexity is very small while the analysis of all possible feasible allocations and feasible attacks can be more costly and it depends on n, x, and N_M. To solve Equations 16-18 we used a straightforward exhaustive search for the optimal solutions: all results have been computed in a few hours on an i9 CPU. Since this kind of evaluation

should be done once for all to design the characteristics of the DSS we believe the computational complexity is acceptable.

7 Related work

Coding based DSS have been introduced to improve a target system performance, e.g. data availability and reliability [10, 12]. Of course, DSS security properties have attracted attention too. In the following we discuss the more closely related works organized in terms of the different aspects they tackle.

Availability and reliability. Most works published in the field leverage on Maximum Distance Separable (MDS), e.g. classical Reed-Solomon codes, where the original data can be retrieved from any random combination of k (out of n) coded fragments. Initially, the homogeneous case where all used SNs store the same amount of fragments has been studied [13]. Recently, a line of research has dealt with the problem of finding the optimal allocation of coded-fragments in DSS where SNs are unreliable with heterogeneous probabilities of responding to queries of a data collector. In [22] an MDS code is considered and various approximations are proposed to find allocations that maximize the probability of decoding a data unit. The authors of [15] propose an efficient calculation of the reliability of an allocation in a heterogeneous DSS by reducing it to linear computation cost that is based on a newly proposed weighted k-out-of-s model. In [23] the authors propose efficient algorithms for optimizing over a few classes of allocations that are derived from the symmetric case where each SN is given the same amount of coded fragments. It is shown by numerical analysis that the proposed allocations outperform existing methods. The work in [24] defines a notion of guaranteed allocations and finds the necessary conditions for an allocation to be guaranteed based on individual SNs capacities and overall storage budget. An iterative algorithm is developed to find guaranteed allocations, if feasible.

The redundancy of DSS can be exploited to recover from losses, with the so called repair mechanism. In [14] the authors study the capacity of heterogeneous DSS where SNs can have different storage capacities and different repair bandwidths. Lower and upper bounds on the system capacity are given as a function of the average resources per SNs. In [3] an overview of the possible approaches for exact and functional repair is provided, showing that using network coding techniques the repair bandwidth can be reduced significantly compared to standard MDS codes. In [25] a coding approach is proposed for both a globally regenerating and a locally repairable coding scheme. In [10, 11] erasure codes achieving optimal trade-off between repair bandwidth and storage overhead are proposed. In [12] the benefit of repairable codes is experimentally analyzed in the context of large-scale storage systems.

Security. When resorting to DSS one does not want to compromise on security. The readers can refer to [26] for a recent overview and a detailed taxonomy of the existing approaches to counteract passive attacks in the broad area of network coding systems. As far as DSSs are regarded, several papers have

dealt with the problem of securing from eavesdropper, perform integrity check on collected data and identification of malicious SNs in presence of pollution attack.

Cryptographic or algebraic verification techniques have been proposed by several papers, e.g., [27, 28, 16, 29, 30, 31, 32, 33, 34, 35, 36, 37]. These approaches are hindered by the significant communication overhead required for distribution of verification information and computational costs for data verification. In [38] a Luby transform based cloud storage service is proposed, where hashing is used to define retrieval and verification tags that are appended to each coded fragment. Such tags support exact repair and data verification by allowing a third party to perform the public integrity verification.

Another important approach to deal with pollution attacks to DSS is error correction of corrupted fragments [39, 40, 41, 42, 43]. All these methods exploit error correcting codes to detect fragment corruption and to recover the original data but they may remarkably increase the overall coding overhead; indeed error correction capacity of ideal linear codes is one half the corresponding error detection capacity. For this reason in our work we exploit linear code for detection only and let the proposed identification algorithm progressively recognize and remove corrupted fragments.

A few studies are closer to our contribution. Indeed, they mainly focus on the identification of malicious storage nodes neither by resorting to any external verification entity, nor by exploiting fragment signatures.

In [44] 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. As opposed to ours, [44] relies on random linear coding in Galois Field of very large size so as to guarantee that exactly $k+1$ coded fragments are enough to detect pollution. Another major difference is that the coded fragment allocation problem, which is one of the main contributions of our work, is not considered. Indeed, in [44] each data unit is allocated to n storage nodes, each holding exactly one coded fragment.

In [45] we devised a pollution identification mechanism based on statistical inference tailored to the DSS we proposed in [46]. In [45] we exploit the so called Belief Propagation algorithm that only provides an estimate of the probability of a SN being malicious, while in this work we propose MIND and show that it provides the correct identification always almost surely. Furthermore, allocation of coded fragments is not dealt with because an homogeneous allocation of coded fragments to SNs is assumed as in [44]. In this work we overcome the limitations of the identification mechanism in [45] allowing us to analyze a much more general scenario where coded fragments can be arbitrarily placed on SNs. Finally, in [45] the performance of the Belief Propagation algorithm is assessed only by simulation while in the current work we develop an extremely accurate mathematical model that is able to predict what is the hit probability of MIND.

8 Conclusions

In this paper we studied the problem of identifying malicious nodes in coded distributed storage systems under pollution attacks.

To this end, we developed MIND, an algorithm to identify polluted coded fragments exploiting the notion of virtual storage nodes that are subsets of coded fragments held by actual storage nodes. We formally proved that MIND termination is guaranteed and that when MIND returns a solution it is correct with probability that can be made arbitrarily close to 1 by tuning coding parameters. We further developed a mathematical characterization of the performance of MIND in terms of the probability of finding the set of polluted coded fragments (hit probability), the average number of attempts before succeeding (complexity). We validated the model predictions against results we obtained from a C++ prototype we developed to represent all features of MIND.

As an example of applicability of our work, we exploited the mathematical model to determine optimal allocations in a wide range of values for the system parameters in the case of RLNC encoding. Our findings show that optimal allocations depend in a non trivial way on some parameters of the algorithm as well as on the total amount of malicious SNs in the system.

We plan to further extend our research in several directions:

- the exploration of MIND performance with respect to other encoding techniques for DSS (e.g., systematic codes where coded and non-coded fragments co-exist) as well as for systems where storage nodes can be unreliable and may fail to provide coded fragments to clients.
- The design of efficient decoding algorithms (and the optimization through parallelization of their implementations) to be evaluated in a test-bed.

References

[1] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google File System," in *ACM SOSP*, Oct. 2003.

[2] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *Mass storage systems and technologies (MSST), 2010 IEEE 26th symposium on*. Ieee, 2010, pp. 1–10.

[3] A. G. Dimakis, K. Ramchandran, Y. Wu, and C. Suh, "A survey on network codes for distributed storage," *Proceedings of the IEEE*, vol. 99, no. 3, pp. 476–489, 2011.

[4] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, S. Yekhanin *et al.*, "Erasure coding in windows azure storage." in *Usenix annual technical conference*. Boston, MA, 2012, pp. 15–26.

[5] S. Muralidhar, W. Lloyd, S. Roy, C. Hill, E. Lin, W. Liu, S. Pan, S. Shankar, V. Sivakumar, L. Tang *et al.*, "f4: Facebook's warm blob storage system,"

in *Proceedings of the 11th USENIX conference on Operating Systems Design and Implementation*, 2014, pp. 383–398.

[6] Y. Xiang, V. Aggarwal, Y. R. Chen, and T. Lan, "Differentiated latency in data center networks with erasure coded files through traffic engineering," *IEEE Transactions on Cloud Computing*, vol. 7, no. 2, pp. 495–508, 2019.

[7] E. Bacis, S. De Capitani di Vimercati, S. Foresti, S. Paraboschi, M. Rosa, and P. Samarati, "Dynamic allocation for resource protection in decentralized cloud storage," in *Proc. of the 2019 IEEE Global Communications Conference (GLOBECOM 2019)*, Waikoloa, Hawaii, USA, December 2019.

[8] G. R. Goodson, J. J. Wylie, G. R. Ganger, and M. K. Reiter, "Efficient byzantine-tolerant erasure-coded storage," in *International Conference on Dependable Systems and Networks, 2004*, 2004, pp. 135–144.

[9] K. K. Rao, J. L. Hafner, and R. A. Golding, "Reliability for networked storage nodes," *IEEE Transactions on Dependable and Secure Computing*, vol. 8, no. 3, pp. 404–418, 2011.

[10] Q. Liu, D. Feng, H. Jiang, Y. Hu, and T. Jiao, "Systematic erasure codes with optimal repair bandwidth and storage," *ACM Transactions on Storage*, vol. 13, no. 3, 2017.

[11] Q. Liu, D. Feng, Y. Hu, Z. Shi, and M. Fu, "High-performance general functional regenerating codes with near-optimal repair bandwidth," *ACM Transactions on Storage*, vol. 13, no. 2, 2017.

[12] O. Kolosov, G. Yadgar, M. Liram, I. Tamo, and A. Barg, "On fault tolerance, locality, and optimality in locally repairable codes," *ACM Transactions on Storage*, vol. 16, no. 2, 2020.

[13] D. Leong, A. G. Dimakis, and T. Ho, "Distributed storage allocations," *IEEE Transactions on Information Theory*, vol. 58, no. 7, pp. 4733–4752, 2012.

[14] T. Ernvall, S. El Rouayheb, C. Hollanti, and H. V. Poor, "Capacity and security of heterogeneous distributed storage systems," *IEEE Journal on Selected Areas in Communications*, vol. 31, no. 12, pp. 2701–2709, 2013.

[15] G. Xu, S. Lin, G. Wang, X. Liu, K. Shi, and H. Zhang, "Hero: Heterogeneity-aware erasure coded redundancy optimal allocation for reliable storage in distributed networks," in *Performance Computing and Communications Conference (IPCCC), 2012 IEEE 31st International*. IEEE, 2012, pp. 246–255.

[16] Z. Yu, Y. Wei, B. Ramkumar, and Y. Guan, "An efficient signature-based scheme for securing network coding against pollution attacks," in *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, 2008.

[17] O. Trullols-Cruces, J. M. Barcelo-Ordinas, and M. Fiore, "Exact decoding probability under random linear network coding," *IEEE communications letters*, vol. 15, no. 1, pp. 67–69, 2010.

[18] V. Bioglio, R. Gaeta, M. Grangetto, and M. Sereno, "On the fly gaussian elimination for lt codes," *IEEE Communication Letters*, vol. 13, pp. 953–955, 2009.

[19] X. Zhao, "Notes on "exact decoding probability under random linear network coding"," *IEEE Communications Letters*, vol. 16, no. 5, pp. 720–721, 2012.

[20] Feng Lu, C. H. Foh, Jianfei Cai, and L. Chia, "Lt codes decoding: Design and analysis," in *IEEE International Symposium on Information Theory*, 2009, pp. 2492–2496.

[21] A. Fiandrotti, V. Bioglio, M. Grangetto, R. Gaeta, and E. Magli, "Band codes for energy-efficient network coding with application to p2p mobile streaming," *IEEE Transactions on Multimedia*, vol. 16, no. 2, pp. 521–532, 2014.

[22] V. Ntranos, G. Caire, and A. G. Dimakis, "Allocations for heterogenous distributed storage," in *Information Theory Proceedings (ISIT), 2012 IEEE International Symposium on*. IEEE, 2012, pp. 2761–2765.

[23] Z. Li, T. Ho, D. Leong, and H. Yao, "Distributed storage allocation for heterogeneous systems," in *Communication, Control, and Computing (Allerton), 2013 51st Annual Allerton Conference on*. IEEE, 2013, pp. 320–326.

[24] M. Noori and M. Ardakani, "Allocation for heterogeneous storage nodes," *IEEE Communications Letters*, vol. 19, no. 12, pp. 2102–2105, 2015.

[25] N. Silberstein, A. S. Rawat, and S. Vishwanath, "Error-correcting regenerating and locally repairable codes via rank-metric codes," *IEEE Transactions on Information Theory*, vol. 61, no. 11, pp. 5765–5778, 2015.

[26] Y. Liu and Y. Morgan, "Security against passive attacks on network coding system – a survey," *Computer Networks*, vol. 138, pp. 57 – 76, 2018.

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

[28] C. Gkantsidis and P. Rodriguez, "Cooperative security for network coding file distribution," in *IEEE INFOCOM*, 2006.

[29] E. Kehdi and B. Li, "Null keys: Limiting malicious attacks via null space properties of network coding," in *INFOCOM 2009, IEEE*.

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

[31] X. Wu, Y. Xu, C. Yuen, and L. Xiang, "A tag encoding scheme against pollution attack to linear network coding," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 1, pp. 33–42, 2014.

[32] F. Chen, T. Xiang, Y. Yang, and S. Chow, "Secure cloud storage meets with secure network coding," in *INFOCOM, 2014 Proceedings IEEE*, pp. 673–681.

[33] A. Le and A. Markopoulou, "Nc-audit: Auditing for network coding storage," in *Network Coding (NetCod), 2012 International Symposium on*, pp. 155–160.

[34] S. T. Shen, H. Y. Lin, and W. G. Tzeng, "An effective integrity check scheme for secure erasure code-based storage systems," *IEEE Transactions on Reliability*, vol. 64, no. 3, pp. 840–851, 2015.

[35] M. Baldi, F. Chiaraluce, L. Senigagliesi, L. Spalazzi, and F. Spegni, "Security in heterogeneous distributed storage systems: A practically achievable information-theoretic approach," in *2017 IEEE Symposium on Computers and Communications (ISCC)*, 2017, pp. 1021–1028.

[36] Y. Hu, Y. Liu, W. Li, K. Li, K. Li, N. Xiao, and Z. Qin, "Unequal failure protection coding technique for distributed cloud storage systems," *IEEE Transactions on Cloud Computing*, pp. 1–1, 2017.

[37] W. Shen, J. Qin, J. Yu, R. Hao, J. Hu, and J. Ma, "Data integrity auditing without private key storage for secure cloud storage," *IEEE Transactions on Cloud Computing*, pp. 1–1, 2019.

[38] N. Cao, S. Yu, Z. Yang, W. Lou, and Y. Hou, "LT codes-based secure and reliable cloud storage service," in *IEEE INFOCOM*, 2012, pp. 693–701.

[39] T. Ho, B. Leong, R. Koetter, M. Medard, M. Effros, and D. Karger, "Byzantine modification detection in multicast networks with random network coding," *Information Theory, IEEE Transactions on*, vol. 54, no. 6, pp. 2798 –2803, june 2008.

[40] S. Jaggi, M. Langberg, S. Katti, T. Ho, D. Katabi, M. Medard, and M. Effros, "Resilient network coding in the presence of byzantine adversaries," *Information Theory, IEEE Transactions on*, vol. 54, no. 6, pp. 2596 –2603, june 2008.

[41] R. Koetter and F. Kschischang, "Coding for errors and erasures in random network coding," *Information Theory, IEEE Transactions on*, vol. 54, no. 8, pp. 3579 –3591, august 2008.

[42] B. Chen, R. Curtmola, G. Ateniese, and R. Burns, "Remote data checking for network coding-based distributed storage systems," in *Proceedings of the 2010 ACM Workshop on Cloud Computing Security Workshop, CCSW '10*, pp. 31–42.

[43] M. Baldi, N. Maturo, E. Montali, and F. Chiaraluce, "Aont-lt: A data protection scheme for cloud and cooperative storage systems," in *2014 International Conference on High Performance Computing Simulation (HPCS)*, 2014, pp. 566–571.

[44] L. Buttyan, L. Czap, and I. Vajda, "Detection and recovery from pollution attacks in coding-based distributed storage schemes," *IEEE Transactions on Dependable and Secure Computing*, vol. 8, no. 6, pp. 824–838, 2011.

[45] C. Anglano, R. Gaeta, and M. Grangetto, "Securing coding-based cloud storage against pollution attacks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 5, pp. 1457–1469, May 2017.

[46] ——, "Exploiting rateless codes in cloud storage systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 5, pp. 1313–1322, May 2015.

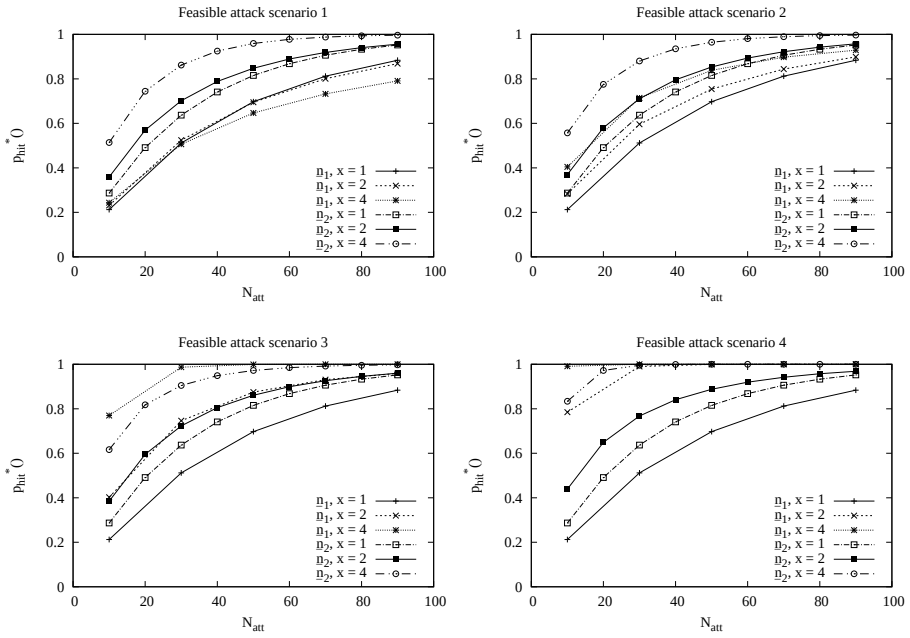

Figure 1: Validation results for $p^*_{hit}()$. Results are for both theoretical results and measures from C++ prototype: lines describe model predictions and points represent C++ prototype output results. Measure points always lie on the corresponding model line.

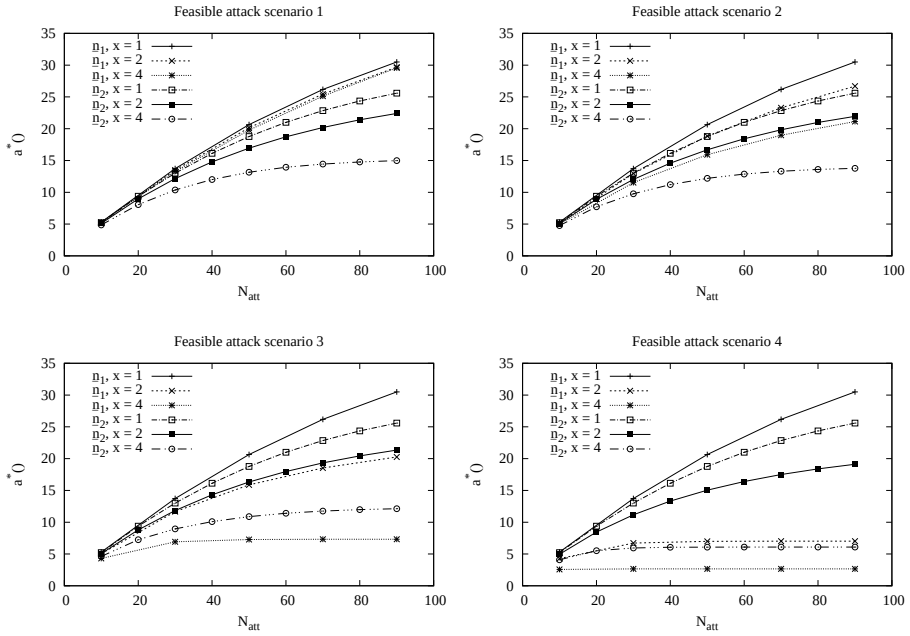

Figure 2: Validation results for $a^*()$. Results are for both theoretical results and measures from C++ prototype: lines describe model predictions and points represent C++ prototype output results. Measure points always lie on the corresponding model line.

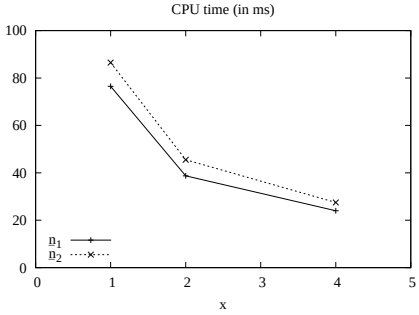

Figure 3: Average running CPU times (in milliseconds) for a successful run of MIND. Results are averaged over the four feasible attacks $\underline{m}_{1,*}$ and $\underline{m}_{2,*}$ for both feasible allocations.

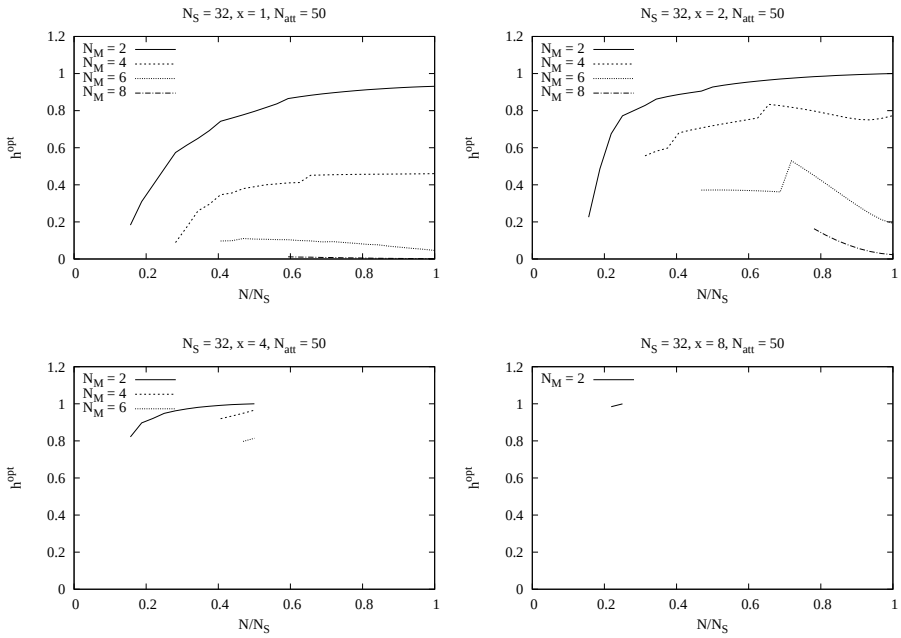

Figure 4: Values of h^{opt} as a function of N_M.

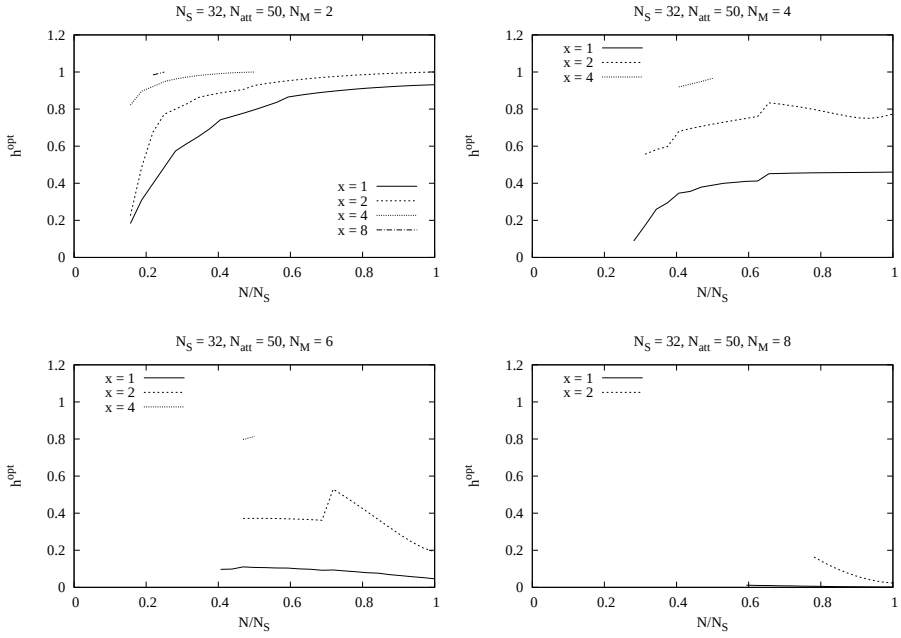

Figure 5: Values of h^{opt} as a function of x.

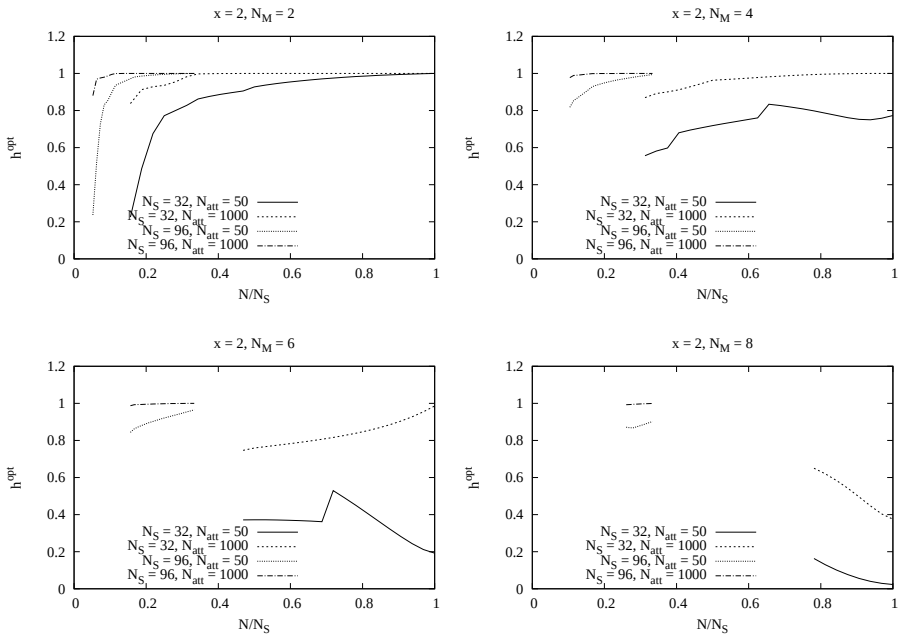

Figure 6: Values of h^{opt} as a function of N_{att} for $N_S = 32$ and $N_S = 96$.