Securing Coding Based Distributed Storage in Wireless Sensor Networks

Levente Buttyán, László Czap, István Vajda
Budapest University of Technology and Economics
Laboratory of Cryptography and Systems Security (CrySyS)
{buttyan,czap,vajda}@crysys.hu

Abstract

We address the problem of pollution attacks in coding based distributed storage systems proposed for wireless sensor networks. In a pollution attack, the adversary maliciously alters some of the stored encoded packets, which results in the incorrect decoding of a large part of the original data upon retrieval. We propose algorithms to detect and recover from such attacks. In contrast to existing approaches to solve this problem, our approach is not based on adding cryptographic checksums or signatures to the encoded packets. We believe that our proposed algorithms are suitable in practical systems.

1 Introduction

In many wireless sensor network (WSN) applications, there are multiple, distributed sources that generate data that must be stored efficiently in multiple storage nodes, each having constrained communication, computation, and storage capabilities. Using the principles of network coding [1, 6, 8, 13] and storing encoded data instead of raw data can help to increase the efficiency of the system. Suppose we have k source nodes and n storage nodes. Instead of storing raw data packets, each storage node stores a linear combination of a subset of the k data packets. The random coding techniques (distributed erasure codes, fountain codes) introduced in [2, 3, 4, 5] ensure that, for appropriately selected parameters, a collector node can reconstruct all the k data packets with high probability by downloading the encoded packets from *any* k storage nodes and solving a system of linear equations (s.l.e.). Thus, the collector node can retrieve the interested data from k nearby nodes, which results in decreased energy consumption, and hence, longer network lifetime. Note that these are primary design criteria in WSNs.

While coding may increase the efficiency of distributed storage systems in a benign environment, it has a potential problem in hostile environments, where an adversary may attack the storage nodes. In particular, the problem that we are interested in in this paper is the so called *pollution attack*, whereby the adversary modifies some of the stored encoded data, which results in erroneous decoding of a large part of the original data upon retrieval. Note that coding schemes mix (typically, linearly combine) blocks of the original data, therefore, a single corrupted encoded block can affect the decoding of multiple data blocks. This amplification effect of the pollution attack is particularly annoying and undesirable.

The latest important result for correcting errors introduced by a Byzantine adversary in network coding based communication systems is presented in [10]. In that paper, the authors introduce an information-theoretically rate optimal code. The packets from the adversarial nodes are intuitively considered as packets coming from a second source, and the packets arriving at the destination are linear combinations of the source's batch of packets and the adversary's batch of packets. Compared to that work, the model we have apopted is different, because we assume k independent sources each producing one packet (per time epoch). In addition, we do not assume any encoding of packets at the source nodes.

Cryptographic techniques have also been proposed to detect attacks in coding based communication and storage systems. An approach to prevent the pollution attack is to require the source nodes to digitally sign [11] (or hash [9]) the data blocks before they are injected in the system. The digital signature scheme must have some homomorphic properties, similar to the case of homomorphic hash functions. Recently, a homomorphic digital signature scheme has been proposed for network coding based content distribution in [12] based on an elliptic curve.

Unfortunately, homomorphic signature schemes are computationally expensive, and they need a public key infrastructure (PKI) for the management of the signature verification keys. These problems hinder their usage in practical applications; in particular, due to the large computational complexity they cannot be used in sensor networks.

Our main contribution in this paper is a novel non-

978-1-4244-2575-4/08/$20.00

cryptographic approach to counteract pollution attacks in coding based distributed storage systems in WSNs. Compared to other approaches in the same vein, we do not add redundancy to the data packets, but rather, we take advantage of the inherent redundancy provided by the coding scheme itself. This redundancy comes from the fact that the content of each storage node corresponds to the same data block vector. To the best of our knowledge, our proposal is the first error detection/correction method that does not require any new functionality at the source nodes or at the storage nodes.

We believe that our proposal is much more practical than the approach based on homomorphic digital signatures. First of all, we need neither a PKI, nor any cryptographic key management scheme, as we do not use cryptography at all. The practical value of this feature should not be underestimated. Second, while our approach also requires intensive computational effort, this is required only for the entity that retrieves information from the distributed storage system. In wireless sensor networks, where the computational overhead really matters, this entity is typically the base station, which is usually assumed to be powerful enough. In contrast to this, in the approach based on homomorphic digital signatures, the source nodes and the storage nodes need to perform intensive computation, and those are typically resource constrained sensor nodes.

The remainder of the paper is organized as follows: In Section 2, we introduce the system model and the adversary model. In Section 3, we describe our proposed attack detection algorithm, together with the analysis of its error probability and complexity. In Section 4, a recovery algorithm is proposed and analyzed. Finally, in Section 5, we draw some conclusions.

2 Model

System model: The general model of the distributed storage systems that we consider in this paper is taken from [3]. The system consists of k source nodes, n storage nodes, and one or more collector nodes. Note that these are roles, and therefore, the sets of source nodes, storage nodes, and collector nodes may overlap.

Each source node i generates a data block X_i, and transfers it to some randomly selected subset of the storage nodes. Each storage node j computes a random linear combination of all the data blocks that it receives; the result is a single code block Y_j. Formally, we can write that $Y_j = XG_j$, where $X = (X_1, X_2, \ldots, X_k)$ is the row vector of all the data blocks, and $G_j = (g_{1j}, g_{2j}, \ldots, g_{kj})^\mathsf{T}$ is a column vector, the non-zero elements of which are the random coefficients used in the linear combination. Here, $g_{ij} \in GF(q)$ for all $i = 1, 2, \ldots, k$ and $j = 1, 2, \ldots, n$, and for some q. Each storage node j stores the pair $Z_j = (G_j, Y_j)$, which represents the equation $Y_j = XG_j$.

The entire system is represented by the system of linear equations (s.l.e.) $Y = XG$, where $Y = (Y_1, Y_2, \ldots, Y_n)$ is the row vector of all code blocks, and $G = (G_1, G_2, \ldots, G_n)$ is a $k \times n$ matrix that contains the coefficient vectors in its columns. Matrix G is also called generator matrix. For appropriately selected values of k and q, any $k \times k$ submatrix of G is non-singular with high probability (see e.g., Theorems 1 and 2 in [3]), therefore, the collector node can reconstruct all the data blocks with high probability by downloading the equations from any k storage nodes and solving the obtained s.l.e. for X. In the rest of the paper, we assume that this property of G holds.

In fact, each data block X_i can itself be a column vector of m symbols $(x_{1i}, x_{2i}, \ldots, x_{mi})^\mathsf{T}$, where $x_{\ell i} \in GF(q)$ for all $i = 1, 2, \ldots, k$ and $\ell = 1, 2, \ldots, m$. In that case, each code block Y_j is also a column vector $(y_{1j}, y_{2j}, \ldots, y_{mj})^\mathsf{T}$ of m symbols in $GF(q)$. The linear combination $Y_j = XG_j$ is computed in a symbol-by-symbol manner, meaning that $y_{\ell j} = \sum_{i=1}^{k} x_{\ell i} g_{ij}$ for all $j = 1, 2, \ldots, n$ and $\ell = 1, 2, \ldots m$. Thus, one can think of X and Y in the s.l.e. $Y = XG$ as matrices of size $m \times k$ and $m \times n$, respectively.

Adversary model: We assume that the adversary has access to t storage nodes, and she can observe and modify the equations stored by them. This means that if the adversary has access to storage node j, then she can modify both G_j and Y_j stored by node j. Let $G^* = G + \Delta G$ and $Y^* = Y + \Delta Y$ be the modified generator matrix and the modified code block vector after an attack, where the modifications made by the adversary are contained in matrix ΔG and vector ΔY. We assume that the adversary modifies the content of each compromised node independently.

Note that the adversary has no access to the source nodes, neither she can attack the communication between the source nodes and the storage nodes. This assumption is plausible, because storage nodes are exposed to attacks for an extended period of time, whereas the source nodes, and the communication between them and the storage nodes must be attacked during the limited time period of data generation and distribution.

The adversary has no information about which k storage nodes will be chosen by the collector node for reconstruction, neither has the collector node information about which storage nodes are compromised. In the sequel, we will assume without loss of generality that the adversary randomly chooses the t storage nodes to be compromised, and the collector node downloads the equations of the first k storage nodes, where the order of the storage nodes is defined randomly by the collector node. Thus, the set of equations downloaded by the collector node is $Z^*_{1..k} = (G^*_{1..k}, Y^*_{1..k})$, where $G^*_{1..k} = (G^*_1, G^*_2, \ldots, G^*_k)$ and $Y^*_{1..k} = (Y^*_1, Y^*_2, \ldots, Y^*_k)$.

Let us now investigate the effect of an attack. The collector node solves the s.l.e. $Y^*_{1..k} = XG^*_{1..k}$ for X, and obtains the result $X^* = Y^*_{1..k}(G^*_{1..k})^{-1}$. The modification induced by the attack in the decoded data blocks can be computed as follows:

$$\begin{aligned} X + \Delta X &= (Y_{1..k} + \Delta Y_{1..k})(G^*_{1..k})^{-1} \\ (X + \Delta X)G^*_{1..k} &= Y_{1..k} + \Delta Y_{1..k} \\ X\Delta G_{1..k} + \Delta X G^*_{1..k} &= \Delta Y_{1..k} \\ \Delta X &= (\Delta Y_{1..k} - X\Delta G_{1..k})(G^*_{1..k})^{-1} \end{aligned}$$

where in the second step we used that $G^*_{1..k} = G_{1..k} + \Delta G_{1..k}$ and $XG_{1..k} = Y_{1..k}$. If at least one of the first k coefficient vectors has been modified by the adversary, then $G^*_{1..k} \neq G_{1..k}$, and thus, $(G^*_{1..k})^{-1}$ can be completely different from $(G_{1..k})^{-1}$. Therefore, in general, such a modification affects all decoded data blocks in every row. Even if the adversary modifies code blocks only and all coefficient vectors are intact ($G^* = G$), all decoded data blocks are affected, however the effect is limited to the rows of X that correspond to a nonzero row in ΔY.

These observations illustrate the amplification effect of the pollution attack: a small amount of modifications in the stored coded information can result in a large amount of modifications in the decoded data. In the worst case all data blocks are entirely destroyed. Below, we address this problem by proposing mechanisms to detect and recover from such attacks.

3 Attack detection

Principle: The basic idea of our attack detection mechanism is the following: We observe that it is very unlikely that the adversary will compromise all the first k equations. Indeed, the probability of this event is around $(t/n)^k$. Thus, some parts of $Y^*_{1..k}$ and $G^*_{1..k}$ are not controlled by the adversary, and for this reason, she cannot enforce a particular solution $X^* = Y^*_{1..k}(G^*_{1..k})^{-1}$. Indeed, X^* will be a random vector in most of the cases, except if all the first k equations are intact, in which case $X^* = X$ will hold.

Now, suppose that we have an additional intact equation: $Y_{k+1} = XG_{k+1}$ (i.e., the collector downloaded $Z_{k+1} = (G_{k+1}, Y_{k+1})$). If X^* is random, then it will not satisfy the additional intact equation with high probability, while it will satisfy it with probability 1 if $X^* = X$. Thus, we can detect if the decoded data block vector X^* is polluted with the help of an additional intact equation.

Algorithm: The proposed attack detection algorithm works in the following way: The collector downloads the first k equations $Z^*_{1..k}$ and computes $X^* = Y^*_{1..k}(G^*_{1..k})^{-1}$. Then, the collector downloads the next equation Z^*_{k+1}. If $Y^*_{k+1} = X^*G^*_{k+1}$, then no attack is detected (and the collector accepts X^* as the correct solution). Otherwise, if $Y^*_{k+1} \neq X^*G^*_{k+1}$, an attack is signalled.

Complexity: We measure the communication complexity in the number of downloaded equations and the computational complexity in the number of s.l.e.'s that we need to solve. Thus, the communication complexity of the proposed attack detection algorithm is $k + 1$, and its computational complexity is 1. As the collector needs to download k equations and solve one s.l.e. in any case, the incurred communication overhead of the attack detection is the theoretical minimum: 1 more equation to download.

Probability of a false negative decision: Let us assume for the moment that the adversary does not modify the coefficient vectors, meaning that $G^* = G$. In this case, the collector obtains the solution $X^* = X + \Delta Y_{1..k}G^{-1}_{1..k} = X + \Delta X$.

If we further assume that the additional equation that we use for detection is intact, then we have $Z^*_{k+1} = Z_{k+1} = (G_{k+1}, Y_{k+1})$. In this case, the false negative error probability, denoted by P_{fneg}, can be computed as follows:

$$\begin{aligned} P_{fneg} &= \Pr\{Y_{k+1} = X^*G_{k+1} | \Delta Y_{1..k} \neq \mathbf{0}\} \\ &= \Pr\{Y_{k+1} = (X + \Delta X)G_{k+1} | \Delta Y_{1..k} \neq \mathbf{0}\} \\ &= \Pr\{\Delta X G_{k+1} = \mathbf{0} | \Delta Y_{1..k} \neq \mathbf{0}\} \end{aligned} \quad (1)$$

where in the last step we used that $Y_{k+1} = XG_{k+1}$.

If $\Delta Y_{1..k}$ has a non-zero element in the i-th row (and $G_{1..k}$ is intact), then ΔX also has some non-zero elements in the i-th row. Otherwise, if the i-th row of $\Delta Y_{1..k}$ contains only zeros, then the i-th row of ΔX contains only zeros too.

We can write the i-th element of $\Delta X G_{k+1}$ as

$$\sum_{\ell=1}^{k} \Delta x_{i\ell} g_{\ell(k+1)} \quad (2)$$

By the argument above, (2) is a non-trivial linear combination of the elements of G_{k+1}. However, the elements of G_{k+1} are chosen randomly, therefore, the probability of (2) being 0 is equal to $1/q$.

>From this, it follows that

$$P_{fneg} = \frac{1}{q^{t'}} \quad (3)$$

where t' is the number of rows in $\Delta Y_{1..k}$ that contain non-zero elements. Clearly, in order to maximize the error probability (and hence minimize the success probability) of the detection, the adversary must make all modifications to the code blocks in a single row[1].

[1] Note that if the code blocks contain standard error detection elements, such as a CRC checksum, then at least 2 rows must be changed by the adversary in every attacked code block. Consequently, in that case, we have that $P_{fneg} \leq 1/q^2$.

Next, we keep the assumption that the adversary does not modify the coefficient vectors (hence $G^* = G$), but we assume that the code block of the additional equation that we use for detection is attacked, meaning that $Z^*_{k+1} = (G_{k+1}, Y^*_{k+1}) = (G_{k+1}, Y_{k+1} + \Delta Y_{k+1})$. In this case, a simple derivation similar to the previous case can be used to arrive to the following result:

$$P_{fneg} = \Pr\{\Delta X G_{k+1} = \Delta Y_{k+1} | \Delta Y_{1..k} \neq \mathbf{0}\} \quad (4)$$

Note that the i-th row of ΔX contains only zeros if the i-th row of $\Delta Y_{1..k}$ contains only zeros. In this case, the i-th element of $\Delta X G_{k+1}$ must be a zero too. Thus, if the i-th element in ΔY_{k+1} is not zero, then the above error probability is 0 (i.e., we can detect the attack even though the additional equation used for detection is not intact). On the other hand, if ΔY_{k+1} contains zeros in every row where $\Delta Y_{1..k}$ contains only zeros, then due to the randomness of G_{k+1}, we get again that $P_{fneg} = 1/q^{t'}$, where t' is the number of rows in $\Delta Y_{1..k}$ that contain non-zero elements.

Finally, let us consider the general case when the adversary may modify both the coefficient vectors and the code blocks, hence $\Delta G \neq \mathbf{0}$ and $\Delta Y \neq \mathbf{0}$. Recall that if $\Delta G_{1..k} \neq \mathbf{0}$, then the solution $X^* = Y^*_{1..k}(G^*_{1..k})^{-1}$ obtained from the first k equations is a random vector. It follows that the equation $Y^*_{k+1} = X^* G^*_{k+1}$ holds with probability around $1/q^m$, and thus

$$P_{fneg} = \Pr\{Y^*_{k+1} = X^* G^*_{k+1} | \Delta G_{1..k} \neq \mathbf{0}\} \approx \frac{1}{q^m} \quad (5)$$

The conclusion of this analysis is that the probability P_{fneg} of false negative detection is maximized if the adversary makes modifications only in a single row of the code block matrix Y and leaves the coefficient matrix G intact. In this case, $P_{fneg} = 1/q$. Hence, if q is chosen sufficiently large (in the order of 2^{60}), then the probability of not detecting a pollution attack is negligible.

Probability of a false positive decision: Let us close this section with the analysis of the probability of a false positive decision. A false positive decision may occur only if the first k equations downloaded by the collector node are intact ($Z^*_{1..k} = Z_{1..k}$) and the additional equation downloaded for attack detection is not intact. From this, a good approximation of the probability of a false positive decision, denoted by P_{fpos}, is the following:

$$P_{fpos} \approx \Pr\{\Delta Z_{k+1} \neq \mathbf{0} | \Delta Z_{1..k} = \mathbf{0}\} \quad (6)$$

Given that the first k equations are intact, the probability that the $(k+1)$-st equation is also intact is

$$\frac{\binom{n-k-1}{t}}{\binom{n-k}{t}} = \frac{n-k-t}{n-k} \quad (7)$$

where t is the number of randomly chosen storage nodes that are attacked by the adversary. From this, we get that

$$P_{fpos} = 1 - \frac{n-k-t}{n-k} = \frac{t}{n-k} \quad (8)$$

While P_{fpos} is not negligible, false positive decisions do not have serious effects. Indeed, when the attack detection algorithm signals an attack, the recovery procedure described in the next section is executed. This procedure tries to recover the original data block vector, and as we will see, it succeeds in a few steps when the number of attacked equations is small.

4 Recovery from attack

Principle: When the collector node detects that the originally downloaded set $S = Z^*_{1..k}$ of equations is polluted, it can download more equations and use them to *clean* the polluted set S. The basic idea of cleaning is the following: Let us denote the set of equations downloaded for cleaning by C, and let e be an additional equation. We use the equations in C to replace a subset of size $\leq |C|$ of the equations in S. We denote the resulting new set of equations by S'. Then, we run our attack detection mechanism on S' with equation e used for testing. If no attack is detected, then we accept the solution of the s.l.e. determined by S' as the correct data block vector. Otherwise, we take S again, replace another subset of size $\leq |C|$ of its equations, and run the attack detection again. We repeat these steps until either the cleaning succeeds or all possible replacements of subsets of C have been tried.

In the rest of this section, we propose a specific recovery algorithm based on the principle described above. As we will see, the algorithm is optimal with respect to success probability and communication complexity. However, the price of this optimality is the increased computational complexity. Nevertheless, the algorithm is still usable for practical systems.

We also developed other approaches for recovery purposes. We designed an algorithm that has improved computational complexity, but it has higher communication complexity and it is prepared to clean at most a pre-defined number of attacked equations in S. Due to space limitations we do not detail that algorithm here.

Algorithm: The basic idea of our algorithm is to start the cleaning with a cleaning set C of size one (i.e., to assume first that there is only one attacked equation in set S), and then, if cleaning fails, to increase the size of C iteratively. In this way, sooner or later, we arrive to a cleaning set C that contains as many intact equations as the number of attacked equations in S. In each iteration, we select all possible subsets of the equations in C and replace with them all possible

subsets of equations in S. Thus, eventually, we replace the attacked equations with the intact ones, and arrive to a clean set.

The operation of the algorithm is the following: The algorithm first downloads $Z^*_{1..k+1}$ and runs the attack detection algorithm on $Z^*_{1..k}$ using Z^*_{k+1} as the testing equation. If no attack is detected, then $Z^*_{1..k}$ is clean and the algorithm stops. Otherwise, the algorithm starts the cleaning of $S = Z^*_{1..k}$. This is an iterative process, where in each iteration, exactly one new equation is downloaded. The newly downloaded equation, denoted by e, becomes the testing equation used for attack detection in the current iteration. The rest of the equations downloaded so far, not counting the equations in S, constitute the cleaning set denoted by C. The algorithm takes every possible subset C' of C, such that $|C'| = \tau$ is not greater than k, and uses the equations in C' to replace τ equations in S in all possible ways. After each replacement, the attack detection mechanism is executed on the resulting set S' of equations using e as the testing equation. If no attack is detected, then S' is clean and the algorithm stops.

Success probability: It is easy to see that the algorithm succeeds iff the number t' of the attacked equations in $S = Z^*_{1..k}$ is smaller than the number of the intact equations in the remaining set $Z^*_{k+1..n}$. On the one hand, if this condition holds, then we have at least $t'+1$ intact equations in $Z^*_{k+1..n}$, and therefore, as we continue downloading more and more equations for cleaning, we eventually reach a state where the cleaning set C contains at least t' intact equations and the last downloaded equation e used for attack detection is also intact. In this case, eventually, all the attacked equations in S will be replaced by intact equations from C, hence S will be cleaned. In addition, as e is intact, the attack detection mechanism will indicate no attack, and we can actually realize that S is cleaned.

On the other hand, if t' is not smaller than the number of the intact equations in $Z^*_{k+1..n}$, then either the cleaning set C contains fewer than t' intact equations, and hence, S cannot be cleaned, or C contains exactly t' intact equations and S can be cleaned, but we have no more intact equation for attack detection purposes, and therefore, we cannot realize that S is cleaned.

Given that there are t attacked equations all together, and t' of them are in $Z^*_{1..k}$, we get that the number of intact equations in $Z^*_{k+1..n}$ is $(n-k)-(t-t')$. Hence, the algorithm succeeds iff $t' < (n-k)-(t-t')$, or equivalently, $t < n-k$. Thus, we get that

$$P_{success} = \begin{cases} 1 & \text{if } t < n-k \\ 0 & \text{otherwise} \end{cases} \quad (9)$$

Note that if $t \geq n-k$ then it is theoretically impossible to recover from an attack, hence, our algorithm is optimal with respect to success probability.

Communication complexity: Recall that we measure the communication complexity in the number of the downloaded equations. As the algorithm downloads a new equation in every iteration, its communication complexity depends on the number of the iterations it performs. More precisely, if the algorithm performs R iterations, then its communication complexity is $(k+1)+R$, because it downloads $k+1$ equations at the beginning before the iterative phase is started. As k is a fixed parameter, we are interested in the characterization of R.

The algorithm stops as soon as the following two conditions hold: (a) the number of intact equations in the cleaning set C is equal to the number of attacked equations in S, and (b) the last downloaded equation e used for attack detection is intact. Thus, R is the number of equations needed to be downloaded to satisfy the two conditions above.

It must be clear that if S contains t' attacked equations, then $C \cup \{e\}$ must contain at least $t'+1$ intact equations, as otherwise, we cannot clean S and realize that it has been cleaned at the same time. Thus, R is minimal in the sense that for $R' < R$ downloaded equations, $C \cup \{e\}$ contains fewer than $t'+1$ intact equations, and hence, the algorithm cannot succeed. This means that our algorithm is optimal in terms of communication complexity.

We give an estimation of R in the following way. Let $p = t/n$, and let W_1 denote the number of equations that need to be downloaded in order for the downloaded set of equations to contain exactly the same number of intact equations on average, as the number of attacked equations in S. The average number of attacked equations in set S is approximately kp. The average number of intact equations among the W_1 equations is approximately $W_1(1-p)$. Hence, we get that $W_1 \approx kp/(1-p)$. Furthermore, let W_2 denote the average number of equations that need to be downloaded until we download an intact equation. Clearly, $W_2 \approx 1/(1-p)$. Thus, when W_1+W_2 equations are downloaded, both conditions (a) and (b) are satisfied. In other words, a good estimate of R is $R \approx W_1 + W_2 \approx \frac{kp+1}{1-p}$

Computational complexity: Recall that we measure the computational complexity in the number of s.l.e.'s that need to be solved. In our case, each call to the attack detection algorithm requires the solution of an s.l.e.

For the derivation of the average case computational complexity $\mathcal{P}_{avg}$, we assume that the number of the attacked equations in S is t', where the average value of t' is kt/n. We make the following observations:

- All but the last iterations of the algorithm execute fully. (term (10) in the sum below)

- In the last iteration, the loops that try to clean S with $\tau < t'$ equations from C also execute fully. (term (11) in the sum below)
- When we use $\tau = t'$ equations from C for cleaning, we have to process on average half of the possible selections of t' equations from C until we end up with the subset that contains the t' intact equations of C. For all those selections, the inner loop executes fully and we must process all the possible selections of t' equations from S. (term (12) in the sum below)
- Finally, when we select the subset of C that contains the t' intact equations, we have to process on average half of the possible selections of t' equations from S until we end up with the t' attacked equations of S. (term (13) in the sum below)

Thus, we get that

$$\mathcal{P}_{avg} \approx \sum_{w=1}^{R-1} \sum_{\tau=1}^{\min(w,k)} \binom{w}{\tau} \binom{k}{\tau} + \quad (10)$$

$$\sum_{\tau=1}^{t'-1} \binom{R}{\tau} \binom{k}{\tau} + \quad (11)$$

$$\frac{1}{2} \binom{R}{t'} \binom{k}{t'} + \quad (12)$$

$$\frac{1}{2} \binom{k}{t'} \quad (13)$$

Figure 1 shows the average computational complexity of the recovery algorithm as a function of the number t of attacked equations. The different curves belong to different values of n and k, and the computation is based on the formula given above. Note the logarithmic scale of the y axis.

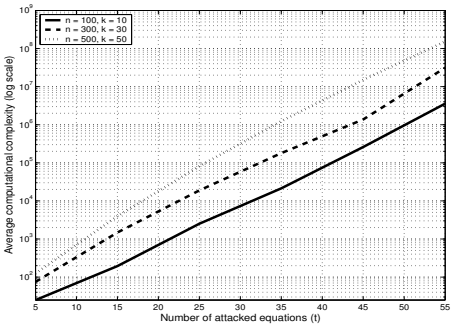

Figure 1. Average computational complexity of the recovery algorithm as a function of the number t of attacked equations. The different curves belong to different values of n and k.

As we noted before, the price of the optimality of the success probability and the communication complexity is the increased computational complexity. However, for practical system sizes of k in the range of 10 to 50 and n in the range of 100 to 500, our algorithm is still feasible in terms of computational complexity. For instance, if $k = 50$, $n = 500$, and $t = 55$, then the computational complexity is approximately $2 \cdot 10^8 \approx 2^{28}$, which is feasible. As another example, consider $k = 10$ and $n = 100$. In this case, successful recovery is ensured even if more than half of the storage nodes are compromised ($t = 55$), with a computational effort of approximately $4 \cdot 10^6 \approx 2^{22}$.

Furthermore, note that the algorithm requires solving a series of s.l.e.'s that differ only in a few equations. This property can be exploited to accelerate the solution of the s.l.e.'s. For details, we refer the reader to the Appendix.

Extension of the algorithm: In this paper we considered an attacker which compromises storage nodes independently. The recovery algorithm can be extended to a more general adversary while retaining its principles. Such an adversary collects the content from all the compromised nodes, processes the obtained information, and according to the output of this processing she reloads the compromised nodes. It is not hard to see that, in this case, the theoretical limit for successful recovery from the attack is $t < n/2$. This limit is achieved by our extended algorithm. Due to space limitations here we give only the main idea of the extension. First, the presented recovery algorithm is executed. The next step is a checking for consistency. If the output of the execution (k output data blocks) is consistent with at least $n/2$ downloaded equations then we can be sure that the output is not compromised. Otherwise, we drop all the equations which are consistent with the output, and execute the algorithm recursively over the remaining set of equations.

5 Conclusion

In this paper, we addressed the problem of pollution attacks in coding based distributed storage schemes in WSNs, and we proposed specific algorithms for detecting and recovering from such attacks. A salient feature of the proposed algorithms is that they are not based on cryptographic checksums or digital signatures, which are traditionally used for providing integrity services. Instead, we take advantage of the inherent redundancy in such distributed storage systems. In addition, our approach does not require the storage nodes to perform additional coding on or to add additional information to the encoded packets. Only the collector node needs to perform a substantial amount of computation. For this reason, we believe that our approach is particularly suitable for wireless sensor networks, where the storage nodes are energy constrained sensors, while the collector is a powerful base station. Detailed comparison with

cryptographic approaches is part of our future work.

While we presented our approach in the context of WSNs, it is, in fact, general, and can be applied in any coding based distributed storage systems, in particular, in the domain of P2P file distribution [7].

Acknowledgements The work presented in this paper has been partially supported by the Mobile Innovation Center (www.mik.bme.hu). Levente Buttyán has been further supported by the Hungarian Academy of Sciences through the Bolyai János Fellowship.

References

[1] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung. Network information flow. *IEEE Transactions on Information Theory*, 46(4):1204–1216, July 2000.

[2] A. G. Dimakis, V. Prabhakaran, and K. Ramchandran. Distributed data storage in sensor networks using decentralized erasure codes. In *Proceedings of the Asilomar Conference on Signals, Systems, and Computers*, Pacific Grove, CA, USA, November 2004.

[3] A. G. Dimakis, V. Prabhakaran, and K. Ramchandran. Ubiquitous access to distributed data in large-scale sensor networks through decentralized erasure codes. In *IPSN '05: Proceedings of the 4th international symposium on Information processing in sensor networks*, page 15, Piscataway, NJ, USA, 2005. IEEE Press.

[4] A. G. Dimakis, V. Prabhakaran, and K. Ramchandran. Decentralized erasure codes for distributed networked storage. *IEEE/ACM Trans. Netw.*, 14(SI):2809–2816, 2006.

[5] A. G. Dimakis, V. Prabhakaran, and K. Ramchandran. Distributed foutain codes for networked storage. In *Proceedings of the IEEE Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Toulouse, France, 2006.

[6] C. Fragouli, J.-Y. L. Boudec, and J. Widmer. Network coding: an instant primer. *SIGCOMM Comput. Commun. Rev.*, 36(1):63–68, 2006.

[7] C. Gkantsidis and P. R. Rodriguez. Network coding for large scale content distribution. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 4, pages 2235–2245, March 2005.

[8] T. Ho, R. Koetter, M. Medard, D. R. Karger, and M. Effros. The benefits of coding over routing in a randomized setting. In *Information Theory Symposium (ISIT). IEEE*, June 2003.

[9] T. Ho, B. Leong, R. Koetter, M. Medard, M. Effros, and D. Karger. Byzantine modification detection in multicast networks using randomized network coding. In *Proceedings of the 2004 IEEE International Symposium on Information Theory (ISIT)*, June 2004.

[10] S. Jaggi, M. Langberg, S. Katti, T. Ho, D. Katabi, and M. Medard. Resilient network coding in the presence of byzantine adversaries. In *Proceedings of the IEEE INFOCOM Conference*, pages 616–624, Anchorage, Alaska, USA, 2007.

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

[12] K. E. Lauter, D. Charles X, and K. Jain. Digital signature for network coding, May 2007.

[13] S. Y. R. Li, R. W. Yeung, and N. Cai. Linear network coding. *Information Theory, IEEE Transactions on*, 49(2):371–381, 2003.

Appendix: During the recovery algorithm a large number of s.l.e.'s must be solved, hence the efficiency of the applied algorithm effects the computational time of the recovery from an attack. We propose a method for accelerating this step. Note that our algorithm replaces a subset of the original set S of linear equations, thus when the size of the substituted set is much smaller than $|S|$, the majority of the equations does not change. We take advantage of this fact.

With set S of the original equations we have $X^*G^*_{1..k} = Y^*_{1..k}$. Let us construct the QR decomposition of $G^*_{1..k}$. $Q_G R_G = G^*_{1..k}$, where Q_G is orthogonal and R_G is triangular. Now we have $X^*Q_G R_G = Y^*_{1..k}$. Solve this s.l.e. for X^*Q_G. As a result of the triangular property of R_G, solving this s.l.e. with back substitution is much more effective than solving a general s.l.e., that requires matrix inversion. By multiplying the result with $Q_G^{-1} = Q_G^{\mathsf{T}}$, we get X^*. When solving the original s.l.e., we perform the costly operation of QR decomposition, but hereafter solving an additional s.l.e. requires only the back substitution after some simple additional computing. This can be done in the following way.

Let us first consider only one equation to replace. Assume we replace the i-th equation with $X'_i G'_i = Y'_i$. Now we have $(X^*)'Q_G R'_G = (Y^*_{1..k})'$. We get R'_G by replacing the i-th column of R_G with $Q_G^{\mathsf{T}} X'_i$, $(X^*)'$ equals X^* with its i-th element replaced with X'_i and similarly $(Y^*_{1..k})'$ is the same as $Y^*_{1..k}$ with its i-th element replaced with Y'_i. We solve this s.l.e. with back substitution to get $(X^*)'Q_G$. This can be done after correcting the triangular property of R'_G. In R'_G only the replaced column interferes this property. By performing a single step of the Gaussian elimination on that column, we get a triangular matrix and can run the back substitution algorithm. Here an additional multiplication results $(X^*)'Q_G Q_G^{\mathsf{T}} = (X^*)'$, that is the solution of the modified s.l.e.

If $c > 1$ equations are replaced, that means c replaced column in R_G, and c steps to perform the Gaussian elimination. The whole method requires $c + 1$ vector-matrix multiplication, c steps of the Gaussian elimination and performing the back substitution algorithm. The overall cost ($O(k^2)$) is lower than the cost of a matrix inversion ($O(k^3)$).

The performance of the recovery algorithm can be further improved approximately by a factor of 2, if the solutions of all the solved s.l.e.'s can be stored.