An Efficient Keyless Fragmentation Algorithm for Data Protection

Katarzyna Kapusta, Gerard Memmi, and Hassan Noura
Télécom ParisTech, Université Paris-Saclay
23, avenue d’Italie,
75013 Paris, France
Email: {firstname.lastname}@telecom-paristech.fr

Abstract—The family of Information Dispersal Algorithms is applied to distributed systems for secure and reliable storage and transmission. In comparison with perfect secret sharing it achieves a significantly smaller memory overhead and better performance, but provides only incremental confidentiality. Therefore, even if it is not possible to explicitly reconstruct data from less than the required amount of fragments, it is still possible to deduce some information about the nature of data by looking at preserved data patterns inside a fragment. The idea behind this paper is to provide a lightweight data fragmentation scheme, that would combine the space efficiency and simplicity that could be find in Information Dispersal Algorithms with a computational level of data confidentiality.

I. INTRODUCTION

Fragmenting data and dispersing them over different physical locations through several transmission paths slows down an attacker from obtaining the totality of the original data. Data are usually transformed into fragments by the use of secret sharing [26], information dispersal algorithms [21], or data shredding [5], [9].

The choice of the most appropriate fragmentation method depends on the particularity of a given use case. A user has to balance between memory use, performance and desired confidentiality level. Fragments obtained with perfect secret sharing are highly secure, but the technique is slow and very costly in memory. Information dispersal is resilient and relatively fast, but not secure. Combining symmetric encryption with fragmentation is secure and easily scalable, but in some circumstances may be less efficient than information dispersal [23].

A fragmentation technique for long-term archival storage of large data will usually differ from the one applied to disperse small data packets. In the last case, the choice would be to pick some lightweight fragmentation technique, like information dispersal or a fast computational secret sharing scheme, rather than to apply more complex mechanisms [12], [18].

The idea of an Information Dispersal Algorithm (IDA) was first introduced by Rabin in the late 80s [21]. Broadly, the algorithm multiplies data chunks by a matrix in order to obtain fragments that are a linear combination of the chunks’ data and matrix elements. The recovery is only possible when a certain amount of these fragments is being gathered. Rabin’s IDA has several advantages: it adds resilience to data, produces almost no storage overhead and uses simple arithmetic operations. Although, the scheme guarantees only incremental confidentiality. An eavesdropper knowing the dispersal matrix can verify if a fragment has a predetermined value. Moreover, such attacker can guess the content of missing fragments when data have recognizable patterns. Despite of such problems, IDAs are still taken into consideration in the context of data protection, as the obstacle of not being able to explicitly reconstruct initial data from less than required amount of fragments may be sufficient in some scenarios [2]. Such scenarios include all use cases were memory or performance overhead caused by secret sharing or cryptographic operations may be a burden and data protection requirements are not too high. An example could be a fragmented database [1], [7] where we would like to quickly retrieve records or a multipath transmission where we would like to defragment only the next destination address [12].

A new scheme situated between computational secret sharing and information dispersal algorithms was recently introduced in a three pages poster [11]. Its complexity and storage overhead are comparable to the one of the IDAs, but it provides higher level of data confidentiality. This paper significantly extends the previous proposal. It contains detailed descriptions, as well as presents more experimental security and performance analyses realized on industrial data.

This paper is organized as follows. Section II summarizes the notation used. Section III contains a description of our contribution. Section IV presents related works. Section V describes the scheme. Section VI presents its empirical security evaluation. Section VII contains a cryptanalysis discussion. Section VIII shows performance results. An insight into future works ends the paper.

II. NOTATION

In order to unify descriptions, we are introducing the notation presented in Table I.


**TABLE I: Notations**

| Symbol | Description |
|--------|-------------|
| $d$    | initial data |
| $d_{size}$ | size of the initial data |
| data chunk | one or more consecutive bytes of original data |
| data share | an encoded data chunk |
| fragment | a final data fragment composed of data shares and stored in one location |
| $k$ | number of fragments required for data recovery |
| $l$ | number of data chunks inside original data |
| $n$ | total number of fragments |
| DCS | Data Chunk Set, a set $k$ of data chunks |
| DSS | Data Share Set, a set $k$ of data shares |
| $S$ | seed, a set of $k$ pseudorandom values |

## III. Our Contribution

The proposed algorithm fragments initial data $d$ of size $d_{size}$ into $n$ fragments of a size close to $\frac{d_{size}}{k}$, any $k$ of which are needed for data recovery. Initial data are processed by sets of $k$ chunks. Data encoding is not based on a matrix multiplication, but on a modification of Shamir’s secret sharing scheme [26] and depends on the encoding results of the previously processed $k$ data chunks. A pseudo-random seed is used as the first set of data chunks and dispersed within the data.

The main benefits of such scheme are:

- No symmetric key encryption is applied, the processing makes use only of simple operations.
- Data patterns are not being preserved inside fragments and the content verification is not straightforward (in contrast to IDAs).
- Partial data defragmentation is possible (in contrast to schemes based on the all-or-nothing transform).

## IV. Related works

This section presents most relevant works from the domains of secret sharing and information dispersal.

### A. Information Dispersal Algorithms

An Information Dispersal Algorithm [21] divides data $d$ into $n$ fragments of size $\frac{d_{size}}{k}$ each, so that any $k$ fragments suffice for reconstruction. More precisely, $n$ data fragments are obtained by multiplying initial data by a $k \times n$ nonsingular generator matrix. Recovery consists in multiplying any $k$ fragments by the inverse of a $k \times k$ matrix built from $k$ rows of the generator matrix. Information dispersal adds redundancy to data and does not produce storage overhead. In [14], Li analyzed the confidentiality of IDAs. For instance, Rabin’s IDA proposal was evaluated to have strong confidentiality, as the original data cannot be explicitly reconstructed from fewer than the $k$ required fragments. However, even if it is not possible to directly recover the data, some information about the content of the initial data is leaked. Indeed, data patterns are preserved inside the fragments when the same matrix is reused to encode different data chunks. A similar problem occurs while using the Electronic Code Book block cipher mode for block cipher symmetric encryption [8].

### B. Shamir’s secret sharing

Shamir’s perfect secret sharing scheme [26] takes as input data $d$ and fragments them into $n$ fragments $f_1, ..., f_n$, of which at least $k$ are needed for data recovery. The algorithm is based on the fact that given $k$ unequal points $x_1, ..., x_k$ and arbitrary values $f_1, ..., f_k$ there is at most one polynomial $y(x)$ of degree less or equal to $k - 1$ such that $y(x_i) = f_i, i = 1, ..., k$. The algorithm provides with the highest level of confidentiality, but has quadratic complexity in function of $k$ and produces fragments of size equal to the initial data. Therefore, it is usually applied for protection of smaller data like encryption keys. In such a use case, drawbacks of the scheme are acceptable, but for larger data they may be a major obstacle.

### C. Secret Sharing Made Short

Krawczyk’s Secret Sharing Made Short (SSMS) [13] combines symmetric encryption with perfect secret sharing for protection of larger data. Data $d$ are encrypted using a symmetric encryption algorithm, then fragmented using an Information Dispersal Algorithm. The encryption key is fragmented using a perfect secret sharing scheme and dispersed within data fragments. In consequence, the solution does not require explicit key management. The storage overhead does not depend on data size, but is equal to the size of the key per data fragment. The performance of the SSMS technique depends on the details of the chosen encryption and IDA techniques.

### D. AONT-RS

The AONT-RS technique [23] is similar to SSMS, as it combines symmetric encryption with data dispersal. It applies an all-or-nothing transform (AONT) [24] to create $k$ fragments: encrypted data are divided into $k - 1$ fragments and an additional fragment is generated by xor'ing hashes of these data fragments with the key used for encryption. Additional $n - k$ fragments are produced using a systematic Reed-Solomon error correction code. Data integrity is ensured by the use of a canary that is dispersed within the fragments.

### E. Parak ́h’s scheme

A steganographic threshold scheme presented in [19] transforms $k$ data chunks into $n$ data fragments using a single polynomial of degree $k - 1$. The size of produced fragments depends on the value of $k$: it decreases while the number of data chunks and the degree of the polynomial are growing.
V. Fragmentation algorithm

This section describes in details the processing core of the proposed fragmentation scheme. Algorithm 1 presents the steps of the fragmentation procedure. Data defragmentation is not presented in the form of an algorithm, as it is basically a direct inverse of fragmentation.

Data processing flow Initial data \( d \) are treated as a concatenation of \( l \) data chunks. These \( l \) data chunks are encoded one by one into \( l \) data shares. Further, \( n \) fragments are constructed from data shares in a way that \( k \) fragments are sufficient for the recovery of \( d \). For a more convenient processing, data chunks are regrouped into Data Chunk Sets of \( k \) elements, where \( DCS_i(j) \) is the \( j \)th data chunk in the set \( i \). Initial data \( d \) may be then presented as a concatenation of \( DCSs: DCS_1,...,DCS_m \) \((m = \lceil \frac{l}{k} \rceil)\). Similarly, data shares are regrouped into Data Share Sets of \( k \) elements, so the result of encoding is a concatenation of \( DSSs: DSS_1,...,DSS_m \). At the end, data shares are distributed to \( k \) final fragments and \( n-k \) redundant fragments are added.

![Diagram](https://via.placeholder.com/150)

Fig. 1: Example for \( k = 3 \): data chunks are transformed into data shares. Encoding of \( DCS_s \) is based on \( DSS_{s-1} \). A pseudorandom seed serves as \( DSS_0 \). \( DSS_1(2) \) comes from a transformation of \( DCS_1(2) \). \( DSS_0(1) \) and \( DSS_0(3) \) were used as coefficients of the encoding polynomial.

Encoding Data processing is done in a Shamir’s like fashion: each data chunk is encoded as a constant term of a polynomial of degree \( k-1 \). More precisely, a data chunk is transformed into a data share inside the function \( Encode \). \( Encode \) takes as parameters the value of the data chunk to be encoded, coefficients \( Coeffs \) of the polynomial, and a point \( x \) at which the polynomial will be evaluated. For each data chunk, \( x \) and \( Coeffs \) are calculated in function of \( DSS_{s-1} \) and \( j \). Therefore, to recover a single data chunk, a user has to possess the \( DSS_i \) containing the result of \( Encode \) for that data chunk, as well as the previous \( DSS_{s-1} \). An example of that is shown in Figure 1. Reusing of data shares trades the perfect security of Shamir’s scheme for a better performance and a smaller size of fragments: during processing the polynomial is evaluated only at one value. Varying not only the polynomial coefficients, but also the values of \( x \), prevents the preservation of data patterns inside encoded fragments.

Seed The \( Encode \) function transforming a data chunk into a data share takes as input \( k-1 \) values of previously encoded data shares. The first set of data chunks \( DCS_1 \) does not possess a predecessor. Thus, a seed composed of \( k \) pseudorandom values is introduced as \( DSS_0 \).

Distributing to fragments After data encoding, data shares are distributed over \( k \) final fragments \( f_1,...,f_k \) inside the \( DistributeShares \) function, in a way that a data share \( DSS_i(j) \) goes to a fragment \( f_j \).

Adding redundancy In a final step \( n-k \) redundant fragments \( f_{k+1},...,f_n \) are added by the function \( AddRedundancy \) implementing a systematic version of a Reed-Solomon error correction code [22].

### Algorithm 1 Fragmentation procedure

\[
\begin{align*}
1. & \quad d = DCS_1,...,DCS_m \\
2. & \quad DSS_0 = s_1,...,s_k \\
3. & \quad \text{for } i = 1 : m \text{ do} \\
4. & \quad \quad \text{for } j = 1 : k \text{ do} \\
5. & \quad \quad \quad x = j \oplus DSS_{i-1}(j) \\
6. & \quad \quad \quad \text{if } x == 0 \text{ then } x = 1 \\
7. & \quad \quad \text{end if} \\
8. & \quad \quad Coeffs = DSS_{i-1}(j) \setminus DSS_{i-1}(j) \\
9. & \quad \quad DSS_i(j) = Encode(DCS_i(j), Coeffs, x) \\
10. & \quad \text{end for} \\
11. & \quad \text{end for} \\
12. & \quad f_1,...,f_k = DistributeShares(DSS_0,...,DSS_m) \\
13. & \quad f_{k+1},...,f_n = AddRedundancy(f_1,...,f_k)
\end{align*}
\]

A. Main characteristics

Complexity Data fragmentation into \( k \) fragments has linear complexity \( O(k) \), as encoding a single data chunk is equal to evaluating a value of a polynomial of degree \( k-1 \) at a single point. Processing redundant fragments depends on the implementation of the error correction code. Same for defragmentation.

Parallelization Defragmentation can strongly benefit from parallelization, as each data chunk is recovered independently from others. Larger data are divided into blocks before applying Algorithm 1 to partially parallelize also the fragmentation processing.

Partial defragmentation An interesting property of the scheme is its fine-grained granularity during defragmentation. Indeed, to defragment a single data chunk it is only required to know its position inside a fragment, as well as possess the previous data share set.

Fragment size The size of produced fragments is close to the space efficient value of \( d_{size}/k \), proposed by Rabin. The only data overhead comes from the seed, which is generated at the beginning of the fragmentation procedure and then attached to data fragments. Seed size depends on the chosen size of the data chunks, as seed values are of the same size than data chunks. Therefore, the fragmentation procedure produces a data overhead of size of one data chunk per fragment.
VI. EXPERIMENTAL SECURITY EVALUATION

An experimental analysis of security characteristics of the scheme based on the methodology presented in [15] was performed. Tests were adapted to the fragmenting nature of the scheme. For instance, we measured the behavior of several parameters (entropy, correlation coefficient, probability density function) in function of number of fragments $k$. A secure fragmentation algorithm should ensure high level uniformity and independence of fragmented data. Section VI-A and VI-B analyze these two properties. Moreover, in Section VI-C we test the sensitivity of the scheme to changes inside the seed.

All tests were performed using Matlab environment on textual data samples provided by LaPoste\(^1\). An example of one of such data sample is shown in Figure 2a. Its corresponding fragment is presented in Figure 2c and compared to the one obtained using an IDA (Figure 2b). Matlab \texttt{rand} function was used for the generation of the pseudorandom seed.

![Figure 2](image-url)

**Fig. 2:** Distribution of a typical textual data sample (a) and distributions of one of its fragments after applying an IDA (b), and after applying the proposed approach (c), for $k=2$. Data patterns are preserved after the use of an IDA. Fragment (c) contains all possible byte values and does not contain visible data patterns. $x$-axis shows the byte position inside the sample, $y$-axis shows the value of the byte at position $x$.

A. Uniformity

Encoded fragments should be characterized by high data uniformity, which is an essential property of a scheme resistant against frequency analysis. We measure fragments uniformity by visualizing their probability density functions, measuring their entropy, as well as applying the chi-squared test.

1) Probability Density Function: Frequency counts close to a uniform distribution testify data have a good level of mixing. This means that each byte value inside a fragment should have an occurrence probability close to $\frac{1}{v} = 0.0039$, where $v$ is the number of possible values (256 for a byte). In Figure 3, the probability density function (PDF) of a data sample and one of its fragment (for $k=2$) are shown. Results for the fragment are spread over the space and have a distribution close to uniform. In Figure 4, the PDF function is also shown, but for different values of $k$ (from 2 to 20). It demonstrates clearly that the occurrence probability of byte values is getting closer to 0.0039 as the value of $k$ is increasing.

![Figure 3](image-url)

**Fig. 3:** Probability density functions of data sample from Figure 2a and its fragment from Figure 2c. $x$-axis shows possible byte values in the sample, $y$-axis shows the probability of occurrence of a value.

![Figure 4](image-url)

**Fig. 4:** Probability density function in function of number of fragments $k$ for different fragments of the data sample from Figure 2a, $x$-axis shows possible byte values in the sample, the $y$-axis shows the probability of occurrence of a value. $k$ value varies from 2 (dark blue line) to 20 (dark red line). With increasing of $k$ the occurrence probability is getting closer to the one of a uniform distribution (0.039).

2) Entropy: Information entropy is a measure of unpredictability of information content [3]. In a good fragmentation scheme the entropy of the fragments should be as high as possible. Figure 5 shows entropy variation for all data fragments compared to entropy variation of original data. The chosen data chunk size is equal to one byte, so the maximum entropy value is equal to 8. The average measured value for overall fragments (7.9926) is significantly higher than the one of original data (5.3498) and close to the maximum. This consequently demonstrates that our scheme ensures the uniformity property.

Furthermore, Figure 6 shows the entropy level in function of the number of fragments $k$. Each entropy value was obtained as an average of entropy of fragmentation results coming from 10 different data samples. The size of one data fragment was set to 1000 bytes. The result shows that the entropy is growing with the number of fragments; it starts

\(^1\)http://www.laposte.fr/
with a values close to 7.91 for a $k = 2$ and achieves 7.99 for values of $k$ close to 20. Moreover, the entropy level is in the same range of values for all $k$ fragments coming from one fragmentation result. This demonstrates that the information is distributed evenly among the fragments.

Fig. 5: Entropy comparison between original and fragmented (with an IDA and with our scheme) data, tested for 100 different data samples, for $k = 5$. y-axis shows entropy values (maximum entropy value is equal to 8), the x-axis the number of the comparison test.

Fig. 6: Variation of entropy in the function of fragments coming from different fragmentation results, $k$ goes from 2 (dark blue dots) to 20 (dark red dots). x-axis shows the fragment’s identifier, y-axis fragment’s entropy value.

3) Chi-squared test: Uniformity of data inside fragments was validated by applying a chi-squared test \cite{6}. For a significance level of 0.05 the null hypothesis is not rejected and the distribution of the fragment data is uniform if $\chi^2_{\text{test}} \leq \chi^2_{\text{theory}}(255,0.05) \approx 293$. The test was applied on fragmentation results of 15 different data samples for a $k$ going up to 20 and fragment size of 1000 bytes. For all values of $k$, the tests was successful. There was no visible correlation between the number of fragments $k$ and the results of chi-squared test.

B. Independence

Fragmented data should be greatly different from its original form. To evaluate fragments’ independence, we analyze recurrence plots, as well as correlation and bit difference between data and their fragments.

1) Recurrence: A recurrence plot serves to estimate correlation inside data \cite{25}. Considering data vector $x = x_1, x_2, ..., x_m$ a vector with delay $t \geq 1$ is constructed $x(t) = x_{1+t}, x_{2+t}, ..., x_{m+t}$. A recurrence plot shows the variation between $x$ and $x(t)$. In Figure 7 such plots for a data sample and its fragments obtained by applying an IDA and the proposed scheme are shown. Using the proposed scheme, no clear pattern is obtained after data fragmentation.

Fig. 7: Recurrence plots for data from Figure 2. x-axis shows data values, y-axis shows data values with a delay $t = 1$. No clear pattern is obtained for a fragment obtained using the proposed scheme.

2) Correlation: Correlation coefficient is used to evaluate the linear dependence between data \cite{25}. A secure fragmentation algorithm should ensure as low correlation as possible between initial data and their fragments.

Correlation coefficients were measured between 10 data samples and their corresponding fragments for different values of number of fragments $k$ (from 2 to 20). The method used for the calculation was same as in \cite{10}. Results are shown in Figure 8. Observed values of correlation coefficients are close to 0. This demonstrates that no detectable correlation exists between tested data samples and their fragments. Moreover, the correlation coefficients for higher values of $k$ tend to have lower values. Then, the correlation between fragments coming from the same fragmentation results was measured. Correlation coefficients among fragments were also close to 0 (in a range of $< -0.01, 0.02 >$). It demonstrated that fragments are not correlated with each other and thus confirmed the independence property of the scheme.

3) Difference: Each fragment should be significantly different from the initial data and from other fragments of the same fragmentation result. Bit difference between a data sample and each of its fragments was measured and it was close to 50%. A similar result was obtained for the difference between fragments themselves.

C. Sensitivity Test

Differential attacks study the relation between two ciphertexts resulting from a slight change, usually of one bit in the original plaintext or in the key. Inspired by this fact, a seed sensitivity test was realized in order to visualize the impact of a change in the seed on the fragmentation result. Indeed, two fragmentation results of same data should be different while obtained using two distinct seeds. For
same data sample of size of 4000 bytes, two fragmentation results were obtained: one for a seed $S_1$ and one for a seed $S_2$ that differs $S_1$ by one random bit. The test has shown that such two fragmentation results are not correlated with each other (correlation coefficients close to 0) and significantly different (around 50% of bit difference).

VII. Cryptanalysis Discussion

In this Section, we discuss about the confidentiality level provided by the proposed algorithm and its resistance against most known types of attack scenarios [4], [10], [17] (statistical, differential, chosen/known plain-text, and brute-force) in a situation when fewer than $k$ fragments have been revealed. During such study, the steps of the algorithm are considered to be public in compliance with Kerckhoff’s principle.

A. General Security Properties

The threat model we use is the one where data fragments are physically dispersed over $n$ different storage locations or transmission paths. Therefore, resulting data protection relies essentially on the difficulty to collect $k$ of the fragments. Indeed, an attacker has to find the locations or transmission paths of dispersed fragments and then manage to access or eavesdrop them. In this paper, we do not deal with data dispersal questions, which includes ensuring the physical separation of fragments, as well as protecting the information describing the order in which fragments have to be assembled to reconstitute the original data. Nevertheless information about the defragmentation procedure should be stored in a secure location or dispersed within the fragments, as its importance could be in a sense compared to the one of the encryption key.

The strength of our solution relies on the use of a seed in form of $k$ pseudo-random values, as well as on unpredictability and high sensitivity of the fragments. Moreover, a user have the possibility to adapt the security level to their needs, not only by increasing the value of number of fragments $k$, but also by augmenting the data chunk size. The proposed scheme ensures forward and backward secrecy, but only in a situation when the seed values are not being repeated.

B. Most known types of attacks

In this Section we are analyzing the resistance of the scheme to the most known types of attacks.

Statistical attacks The category of statistical attacks exploits the fact that encoded data may reveal some statistical properties. Immunity against such attacks requires that fragments achieve high level of randomness [27]. Therefore, in an ideal situation, the frequency analysis of data inside a fragment should be indistinguishable from the output of a pseudorandom generator. Results presented in Section VI have shown that our scheme achieve good uniformity and recurrence characteristics. Indeed, no useful information can be detected from the fragmented data. This demonstrates the high randomness of the scheme and its resistance against statistical attacks. Such property is not achieved by Information Dispersal Algorithms, where in the case of a known generator matrix and data with recognizable patterns it is possible to guess the content of missing fragments.

Brute-force attack and content verification While possessing a $p < k$ number of fragments, an attacker can attempt a brute-force attack by trying to guess the content of the missing $(k - p)$ fragments. Intuitively, the difficulty level of such attack grows with the required number of fragments $k$ and decreases with the number of possessed fragments $p$. The recovery of a set of $k$ data shares of size $w$ each implicates trying $2^w$ possibilities for $k - p$ data shares of missing fragments. Therefore, an increasing of the size of a data chunk/share may harden the brute-force attack on a set of $k$ data chunks. On the other hand, a way of facilitating the attack would be to make some assumptions about the content of the missing fragments that would limit the number of possibilities to verify.

In a different scenario, an attacker with less than $k$ fragments may like to verify whether the data inside fragments matches some predetermined value. In the case of an IDA, such verification is easy when the generator matrix is known. In the case of our scheme, it is harder, as the attacker will have to guess or the missing seed values or the missing data shares used to encode the part of the data they would like to verify.

Known and chosen plain text attacks The knowledge of a part of the plaintext facilitates a brute-force attack. However, as each time a new seed is used, it does not help to recover fragments of other data.

Linear and differential attacks A linear attack consists of exploiting the linear relations between the plaintext, the ciphertext and the key. The knowledge of the first data chunks from the plaintext and of the $p$ fragments allows the recovery of the missing seed values. However,
the recovery of the seed is not as critical, as the key leakage in symmetric encryption schemes. Because fragments are dependent of each others, to decode a data share it is necessary to possess its preceding data shares, also the one that are inside of the $k - p$ missing fragments. In order to avoid differential attacks, seed values have to be change for each fragmentation procedure. A reuse of the seed could expose some relations between fragmentation results, for instance the fragmentation result of two identical plaintexts would be the same. As presented in Section VI-C encoded data show high seed sensitivity, so even a single bit change is sufficient to obtain two different and not correlated fragmentation results of the same plain text.

VIII. Performance Results

Proposed algorithm was implemented in JAVA using the following resources: JDK 1.8 on DELL Latitude E6540, X64-based PC running on Intel® Core™ i7-4800MQ CPU @ 2.70 GHz with 8 GB RAM, under Windows 7. It was tested on data provided by La Poste, the French postal office. An implementation of java.security.SecureRandom is used for seed generation. The scheme can be implemented in any $GF(2^Q)$ and is designed to use only logical operations. For integrity purposes, $Q$ is selected according to word size of processors and can be 8, 16, 32 or 64-bit. As presented in Figure 9 the variation of average time for fragmentation is linear. Similar results were obtained for the defragmentation process. A multi-threaded version, optimized for 4 cores, sped up the performance by a factor of 3 that becomes close to 4 for more intensive computations.

Another test was performed on the tera-memory platform TeraLab using following resources: 32GB RAM and 4 VCPUs. Figure 10 shows the results for fragmentation of a data file of 100MB up to 100 fragments. This result exhibits the linear complexity of the fragmentation relatively to $k$ the number of generated fragments. Figure 11 presents performance measured in function of data size for several values of $k$. Similar results were obtained for the defragmentation process. Results demonstrate that the scheme achieves good scalability.

IX. Future Works

In future, we plan to benchmark our algorithm to compare its performance with most relevant works. A possibility of performance improvement is seen in parallelization of the fragmentation processing by partially limiting the sequential character of the encoding procedure. A more sophisticated way of distribution of encoded data chunks could be also envisioned in order to create fragments taking into account several levels of confidentiality. Such processing would require an additional step in form of the separation of initial data chunks regarding their level of confidentiality. It could be inspired by works done in [20].
that use wavelet transform to separate data into two parts, a private and a public one, without any user interaction. Another, more complex, research track would be to design a complete fragmentation architecture for data protection by means of fragmentation, encryption, and dispersion. This work would focus on secure management of information about the location and order of data fragments, as well as on secure distribution of fragments from the trusted site performing fragmentation procedure to their final storage destinations.

X. Conclusion

A novel fragmentation algorithm for secure and resilient distributed data storage was described and analyzed. It combines the keyless property, computational simplicity and space efficient size of Fragmentation Dispersion Algorithms with an adjustable computational level of security. Security analysis shows that the produced fragments achieve good randomness and are not correlated neither with the initial data, nor among themselves. The use of a fresh seed for each fragmentation procedure ensures the backward and forward secrecy properties, as well as limits the possibility of differential attacks. Brute-force and chosen-plaintext attacks could be considered in some situations. Their attack resistance depends on the number of fragments in the possession of an attacker, as well as the quantity of knowledge about of the initial data.

The scheme was implemented and tested on different samples of industrial data. Tests show good performance and scalability. The fragmentation procedure is linear in terms of number of fragments.

We believe that the scheme could be applied to all application of data storage or transmission, where we would like to hide the nature of fragmented data without applying cryptographic mechanisms.

XI. Acknowledgments

The work is partially funded by the ITEA2 CAP project.

References

[1] A. Blaknie, F. Cuppens, N. Cuppens-Boulahia, and J. M. Fernandez. Confidentiality-preserving query execution of fragmented outsourced data. In Proceedings of the 2013 International Conference on Information and Communication Technology, ICT-EurAsia’13, pages 426–440, Berlin, Heidelberg, 2013. Springer-Verlag.
[2] W. J. Buchanan, D. Lanc, E. Ukwandu, L. Fan, G. Russell, and O. Lo. The future internet: A world of secret shares. Future Internet, 7(4):445, 2015.
[3] C. Cachin. Entropy measures and unconditional security in cryptography, 1997.
[4] J.-S. Cho, S.-S. Yeo, and S. K. Kim. Securing against brute-force attack: A hash-based rfid mutual authentication protocol using a secret value. Comput. Commun., 34(3):391–397, Mar. 2011.
[5] P. Cincilla, A. Boudguiga, M. Hadji, and A. Kaiser. Light blind: Why encrypt if you can share? In 2015 12th International Joint Conference on e-Business and Telecommunications (ICETE), volume 04, pages 361–368, July 2015.
[6] W. G. Cochran. The χ² test of goodness of fit. Ann. Math. Statist., 23(3):315–345, 09 1952.
[7] S. De Capitani di Vimercati, R. F. Ebert, S. Foresti, S. Jajodia, G. Livraga, and P. Samarati. Encryption and Fragmentation for Data Confidentiality in the Cloud, pages 212–245. Springer International Publishing, Cham, 2014.
[8] M. J. Dworkin. Sp 800-38a 2001 edition. recommendation for block cipher modes of operation: Methods and techniques. Technical report, Gaithersburg, MD, United States, 2001.
[9] J.-C. Fabre, Y. Deswarte, and B. Randell. Designing secure and reliable applications using fragmentation-redundancy-scattering: an object-oriented approach, pages 21–38. Springer Berlin Heidelberg, Berlin, Heidelberg, 1994.
[10] Z. Fawaz, H. Noura, and A. Mostefaoui. An efficient and secure cipher scheme for images confidentiality preservation. Image Commun., 42(C):90–108, Mar. 2016.
[11] K. Kapusta, G. Memmi, and H. Noura. Poster: A keyless efficient algorithm for data protection by means of fragmentation. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, CCS ’16, pages 1745–1747, New York, NY, USA, 2016. ACM.
[12] S. Katti, J. Cohen, and D. Katabi. Anonymity using unreliable overlays. In Proceedings of the 4th USENIX Conference on Networked Systems Design & Implementation, NSDI’07, pages 4–4, Berkeley, CA, USA, 2007. USENIX Association.
[13] H. Krawczyk. Secret sharing made short. In Proceedings of the 13th Annual International Cryptology Conference on Advances in Cryptology, CRYPTO ’93, pages 136–146, London, UK, 1994. Springer-Verlag.
[14] M. Li. On the confidentiality of information dispersal algorithms and their erasure codes. CoRR, abs/1206.4123, 2012.
[15] H. Noura, S. Martin, K. A. Agha, and K. Chahine. Erss-rhc: Efficient and robust secure scheme for random linear network coding. Computer Networks, 75, Part A:99 – 112, 2014.
[16] K. Nyberg and L. R. Knudsen. Provable Security Against Differential Cryptanalysis, pages 566–574. Springer Berlin Heidelberg, Berlin, Heidelberg, 1993.
[17] C. Paar and J. Pelzl. Understanding Cryptography: A Textbook for Students and Practitioners. Springer Publishing Company, Incorporated, 1st edition, 2009.
[18] A. Parakh and S. Kak. Online data storage using implicit security. Inf. Sci., 179(19):3223–3331, Sept. 2009.
[19] A. Parakh and S. Kak. Space efficient secret sharing for implicit data security. Information Sciences, 181(2):335 – 341, 2011.
[20] H. Qiu and G. Memmi. Fast selective encryption methods for bitmap images. Int. J. Multimed. Data Eng. Manag., 6(3):51–69, July 2015.
[21] M. O. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. J. ACM, 36(2):335–348, Apr. 1989.
[22] I. S. Reed and G. Solomon. Polynomial Codes Over Certain Finite Fields. Journal of the Society for Industrial and Applied Mathematics, 8(2):300–304, 1960.
[23] J. K. Resch and J. S. Plank. Aont-rs: Blending security and performance in dispersed storage systems. In Proceedings of the 9th USENIX Conference on File and Storage Technologies, FAST’11, pages 14–14, Berkeley, CA, USA, 2011. USENIX Association.
[24] R. L. Rivest. All-or-nothing encryption and the package transform. In In Fast Software Encryption, LNCS, pages 210–218. Springer-Verlag, 1997.
[25] J. L. Rodgers and A. W. Nicewander. Thirteen Ways to Look at Correlation Coefficient. The American Statistician, 42(1):59–66, 1988.
[26] A. Shamir. How to share a secret. Commun. ACM, 22(11):612–613, Nov. 1979.
[27] S. Xu, Y. Wang, J. Wang, and M. Tian. Cryptanalysis of two chaotic image encryption schemes based on permutation and xor operations. In Computational Intelligence and Security, 2008. CIS ’08. International Conference on, volume 2, pages 433–437, Dec 2008.