POSTER: A Keyless Efficient Algorithm for Data Protection by Means of Fragmentation

Katarzyna Kapusta*, Gerard Memmi, and Hassan Noura
Télécom ParisTech, Université Paris-Saclay
46, rue Barrault,
75013 Paris, France
{katarzyna.kapusta, gerard.memmi, hassan.noura}@telecom-paristech.fr

ABSTRACT

Although symmetric ciphers may provide strong computational security, a key leakage makes the encrypted data vulnerable. In a distributed storage environment, reinforcement of data protection consists of dispersing data over multiple servers in a way that no information can be obtained from data fragments until a defined threshold of them has been collected. A secure fragmentation is usually enabled by secret sharing, information dispersal algorithms or data shredding. However, these solutions suffer from various limitations, like additional storage requirement or performance burden. This poster presents a novel flexible keyless fragmentation scheme, balancing memory use and performance with security. It could be applied in many different contexts, such as dispersal of outsourced data over one or multiple clouds or in resource-restrained environments like sensor networks. The scheme has been implemented in JAVA and Matlab. Preliminary analysis shows good performance and data protection.

Keywords

Data protection; Distributed systems; Cloud storage; Fragmentation; Security analysis

1. INTRODUCTION

Moving sensitive and critical data to a not thoroughly trustworthy storage provider forces a user to revisit traditional data protection procedures. With the growing demand for non-costly protection of outsourced data, fragmentation for data protection seems to be a very promising research track. Data fragments distributed over multiple servers or even multiple clouds slow down an attacker who has to get access to different locations.

Fragmentation for security is not a new idea. Shamir introduced it in a seminal paper from the late 70s, where he addressed the problem of secure storage and management of an encryption key [5]. His information-theoretically secure solution fragments data into n fragments, k of which are needed for reconstruction. A decade later, Rabin proposed an information dispersal algorithm (IDA) optimizing fragments size, but lowering protection level [4]. Later, Krawczyk in Secret Sharing Made Short combined both methods, as well as introduced his own IDA [2]. Nowadays, all of these methods are reconsidered in the context of secure data storage in an untrusted distributed environment [1]. Furthermore, Shamir's scheme was an inspiration for several keyless algorithms [3].

The ultimate objective of our algorithm is to combine the space efficiency and keyless property of information dispersal algorithms with a decent level of data protection, while assuring good performance. Our algorithm fragments initial data d of size d_{size} into n fragments of size close to optimal value $\frac{d_{size}}{k}$, any k of which are needed for data recovery. A single fragment does not provide information about initial data.

Besides cloud storage, other applications can benefit from the proposed scheme, including data protection in distributed environments with restrained computational resources and limited storage capacity like unattended wireless sensor networks.

This poster is organized as follows. Section 2 describes our fragmentation algorithm. Section 3 presents an analysis of its security characteristics. Section 4 contains preliminary performance results. An insight into future works ends the poster.

2. FRAGMENTATION ALGORITHM

Fragmentation is described by Algorithm 1. Initial data d are a concatenation of l smaller data chunks. First, these data chunks are transformed into l data shares of same size. Further, n fragments are constructed from such data shares, any k of which are sufficient for the recovery of d.

For a more convenient processing, data chunks are regrouped into Data Chunk Sets of k elements, $DCS_i(j)$ is the jth data chunk in the set i. d may be then presented as $DCS_1, ..., DCS_m$ $(m = \lceil \frac{l}{k} \rceil)$. Each DCS will be encoded into a corresponding Data Share Set $DSS_1, ..., DSS_m$. $DSS_0 = s_1, ..., s_k$ represents the seed and contains k pseudorandom values.

Function *Encode* encodes on a polynomial in a Shamir like fashion, a data chunk $DCS_i(j)$ into a corresponding data share $DSS_i(j)$. As parameters, *Encode* takes the value of the data chunk, coefficients $Coeffs$ of the polynomial, and

*Corresponding author.

CCS'16 October 24-28, 2016, Vienna, Austria

ACM ISBN 978-1-4503-4139-4/16/10.
DOI: http://dx.doi.org/10.1145/2976749.2989043

the point x at which it will be evaluated. x and $Coeffs$ are calculated in function of DSS_{i-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 DSS_{i-1}.

A secure data dispersal is made by the function *Disperse* that allocates data shares to final fragments $f_1, ..., f_k$, in a way that $DSS_i(j)$ goes to fragment f_j. In a final step $n-k$ redundant fragments $f_{k+1}, ..., fn$ are added by the function *AddRedundancy* implementing a version of Reed-Solomon code.

Data defragmentation is the inverse of fragmentation. All operations are done in a finite field that is chosen in function of the desired data chunk size.

Algorithm 1 Fragmentation procedure

1: $d = DCS_1, ..., DCS_m$
2: $DSS_0 = s_1, ..., s_k$
3: **for** $i = 1 : m$ **do**
4: **for** $j = 1 : k$ **do**
5: $x = j \oplus DSS_{i-1}(j)$
6: **if** $x == 0$ **then** $x = 1$
7: $Coeffs = DSS_{i-1} \setminus DSS_{i-1}(j)$
8: $DSS_i(j) = Encode(DCS_i(j), Coeffs, x)$
9: $f_1, ..., f_k = Disperse(DSS_0, ..., DSS_m)$
10: $f_{k+1}, ..., fn = AddRedundancy(f_1, ..., f_k)$
11: Distribute $f_1, ..., f_n$ over n entities

2.1 Complexity and Parallelization

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 has quadratic complexity, but we optimized its implementation to achieve quasi-linear complexity. Same for defragmentation.

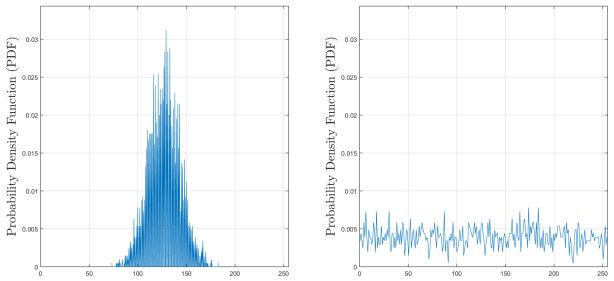
Defragmentation can strongly benefit from parallelization, as each data chunk is recovered independently from others. Larger data are divided into blocks before applying Algorithm 1 to parallelize fragmentation processing.

3. SECURITY ANALYSIS

Cryptographic strength of proposed solution relies on the use of a seed in form of k pseudorandom values, as well as on unpredictability and sensitivity of fragments.

A complete knowledge of fragments belonging to specific data does not permit the recovery of previous or future data, since for each fragmentation procedure k pseudorandom values serve as a seed. Therefore, backward and forward secrecy are assured if the seed is generated in a secure way. A brute-force attack could be considered consisting of finding missing seed values and their corresponding fragments. Its difficulty level grows with the required number of fragments k, as well as their size and decreases with the number of fragments in possession of an attacker. Moreover, fragments are sensitive to changes in the seed: one bit change in the seed results in a fragmentation that is 50% different than the one obtained using the unmodified seed.

In contrary to Rabin's IDA that struggles with the problem of data patterns appearing in fragmented data, our scheme does not preserve distances between encoded data parts (see Section 3.1 and 3.2). Furthermore, it strengthens the Krawczyk's IDA idea by introducing some pseudorandomness



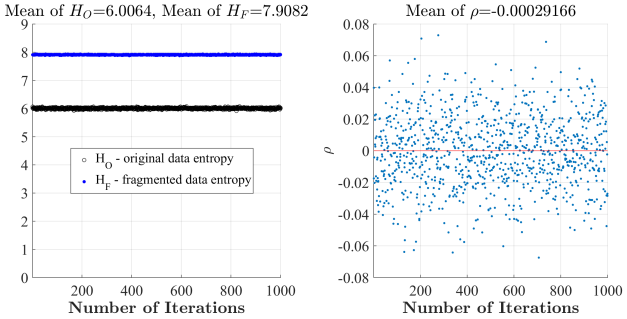
(a) *Original data* (b) *Fragment*

Figure 1: *Probability Density Function, k=8.*

A secure fragmentation algorithm should ensure high level uniformity and independence of fragmented data. Section 3.1 and 3.2 analyze these two properties. During tests, the scheme is considered as a black box and a set of 1024 byte length data to be fragmented is generated, following a normal distribution with mean and standard deviation equal to $(128, 16)$. Matlab *rand* function is used for seed generation.

3.1 Uniformity

To measure uniformity, mixing nature and entropy variation of fragmented data are tested. Frequency counts close to a uniform distribution testify data have a good level of mixing. In Figure 1a and 1b, the Probability Density Function (PDF) of the original data and one of its fragments are shown, respectively. Results for the fragment are spread over the space and have a uniform distribution.



(a) *Entropy* (b) *Correlation*

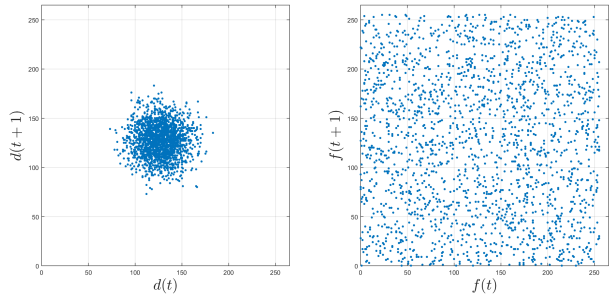
Figure 2: *Entropy comparison between original (H_0, black circles) and fragmented (H_F, blue dots) data. (a) Correlation coefficients between original and fragmented data. (b) (for 1000 times, k=8)*

Figure 2a shows entropy variation for all data fragments compared to entropy variation of original data. Data chunk size is equal to one byte, so the maximum entropy value is equal to 8. The average measured value for overall fragments (7.9082) is significantly higher than the one of original data (6.0064) and close to the possible maximum. This indicates clearly that the scheme ensures the uniformity property.

3.2 Independence

Fragmented data should be greatly different from its original form, as well as the correlation inside data should be as low as possible.

Recurrence test: Recurrence plot serves to measure the evaluation of randomness and estimate correlation inside data. In Figure 3, the recurrence plots of original data and its fragment are shown, respectively. No clear pattern is obtained after fragmentation.



(a) *Original data* (b) *Fragment*

Figure 3: *Recurrence plots, k=8.*

Correlation test: Correlation coefficient is used to evaluate the linear dependence between data. It is measured between original and fragmented data (Figure 2b), and then among 8 different fragments (Figure 4). Values close to zero indicate that no detectable correlation exists between the initial data and its fragment, neither between the fragments. This consequently ensures the independence propriety.

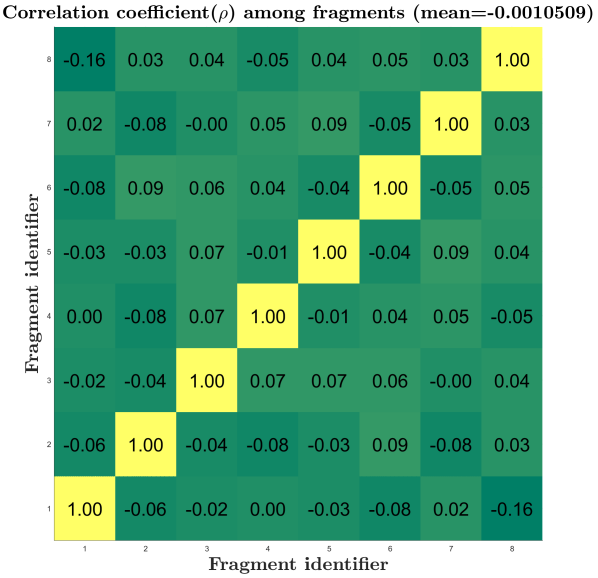


Figure 4: *Correlation coefficient among fragments, k=8.*

4. 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[1]. 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. To ensure better efficiency, Q is selected according to bit size of processors and can be 8,16, 32 or 64-bit. As presented in Figure 5, the variation of average time for fragmentation is **linear**. Same linear results were obtained for a range of k going until $k = 100$. 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.

[1]http://www.laposte.fr/

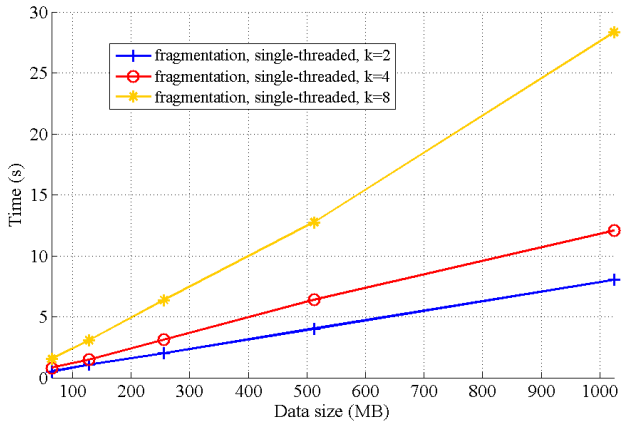


Figure 5: *Variation of the average fragmentation time, single-threaded implementation, $k = 2, 6, 8$.*

5. CONCLUSION

A novel keyless fragmentation algorithm for secure and resilient distributed data storage was introduced and analyzed. It is flexible, produces negligible storage overhead and is easily parallelizable. Its implementation use only logical operations and has good performance and security. Future works consist of benchmarking the scheme, as well as adapting it to two of its uses cases: dispersing massive data in cloud environment and protecting data generated inside sensor networks.

6. ACKNOWLEDGMENTS

This work is partially financed by the ITEA2 CAP project.

7. REFERENCES

[1] K. Kapusta and G. Memmi. Data protection by means of fragmentation in distributed storage systems. In *2015 International Conference on Protocol Engineering (ICPE) and International Conference on New Technologies of Distributed Systems (NTDS)*, pages 1–8, July 2015.

[2] 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, UK, 1994. Springer-Verlag.

[3] A. Parakh. *New Information Dispersal Techniques for Trustworthy Computing*. PhD thesis, Stillwater, OK, USA, 2011. AAI3460014.

[4] M. O. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *J. ACM*, 36(2):335–348, Apr. 1989.

[5] A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, Nov. 1979.