On the Security of a Backup Technique for Database Systems based on Threshold Sharing

Ruxandra F. Olimid, **Dragoş Alin Rotaru**

University of Bucharest, Romania
(e-mail: ruxandra.olimid@fmi.unibuc.ro, r.dragos0@gmail.com)

Abstract: Preserving data privacy is a challenging task, especially when long-term storage and fault-tolerance are required. Long-term storage systems based on secret sharing appear as an alternate solution to the traditional technique of data encryption and backup restore. Alouneh et al. (2013) propose such a method, which they claim to be both secure and efficient. We show their assertion is false and exemplify some vulnerabilities of their construction, which arise natural from the deterministic behavior of their system.

Keywords: data storage; distributed databases; fault tolerance; system security; attacks.

1. INTRODUCTION

As electronic data volume increases daily, the necessity for storage becomes more important. Part of the data is sensitive and hence must be kept secret, sometimes for long periods of time (decades or even centuries). Such data might include classified military information, legal documents or health records: a military file might remain classified for centuries, a will should be kept secret until the owner's death, while the medical record of a patient could be preserved secret even after his death.

Preserving information confidentiality for long time periods is a challenging task due to multiple threats encountered, like large-scale disasters, component faults, attacks or human error (Baker et al. (2006); Storer et al. (2006)). This is especially true nowadays, when organizations tend to move their data in the cloud to decrease the costs required to set up and maintain a data-center; data is thus stored and processed beyond the close control of the owner.

A long-term storage system must satisfy (at least) the following properties:

- *Secure long-term storage.* Data must be available to qualified parties only; it should be infeasible for unauthorized parties to access the data or even partial information about the data through long periods of time (years or decades);
- *Availability.* Data must remain accessible to authorized users even in case of possible failures. Since the data is stored for long periods of time, hardware or software crashes are unavoidable, so the system must provide redundancy to survive such losses;
- *Data integrity.* Data must be certified as original, in the sense that the system attests the data to be unchanged and unmodified.

The traditional solution to provide data confidentiality is encryption. However, encryption is not theoretically secure, being vulnerable to an adversary with unbounded computational power. In practice, guidelines for proper usage of encryption systems are published yearly to prevent usage of vulnerable algorithms or insufficient key length. Thus, an encrypted file can be practically infeasible at the moment is encrypted, but broken in only a few hours or days after 50 years. A possible realization of large-scale quantum computers would make many crypto-algorithms insecure.

An alternative to the traditional solution are storage systems that use *secret sharing*, a cryptographic technique that assures both perfect secrecy and redundancy. Wylie et al. (2000); Subbiah and Blough (2005); Storer et al. (2009) are only a few examples from literature that propose and analyze such solutions.

The current paper analysis the security of the recent proposal of Alouneh et al. (2013), a long-term system that uses Shamir's secret sharing. We show the proposal is insecure: an adversary can disclose some information about the type and content of the stored data. The weaknesses are caused by a flaw in the design: the algorithm is deterministic, hence the construction breaks one of the basic rule of security. For completeness, we indicate the correct way to use secret sharing to avoid determinism and show that practical results validate it.

The paper is organized as follows. Next section introduces secret sharing and briefly reviews the related work on long-term storage systems based on secret sharing. Section 3 compares encryption based and secret sharing based storage systems. Section 4 presents the proposal of Alouneh et al. (2013). Section 5 presents the theoretical vulnerabilities of the system. Section 6 analyzes the practical results. Section 7 reviews the correct usage of secret sharing to avoid the presented vulnerabilities. Last section concludes.

2. SECRET SHARING AND RELATED WORK

Secret sharing is a cryptographic primitive introduced independently by Blakley (1979) and Shamir (1979): a

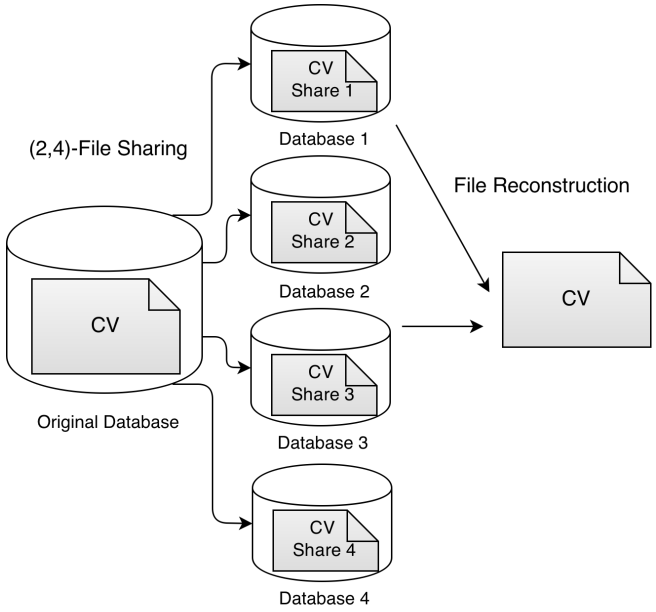

Fig. 1. A $(2, 4)$-secret sharing scheme used for file splitting

secret is split into multiple parts, called *shares* such that later the secret can be recovered from any qualified set of shares. If the scheme is *perfect*, then an unqualified set of shares gives no information about the secret. Hence, even if an attacker gains access to some shares, he learns nothing about the secret data. The particular case of secret sharing that requires at least k out of n shares for reconstruction is called *threshold secret sharing*. Remark that threshold secret sharing provides fault-tolerance by construction for up to $n - k$ unavailable shares.

Fig.1 exemplifies a $(2, 4)$-secret sharing: an original document (e.g.: a curriculum vitae) is split into four shares such that any two shares are enough for reconstruction and a single share leaks no information about the document; the scheme provides fault-tolerance: reconstruction remains possible if one or two shares are lost.

Several systems were developed on top of secret sharing to provide survivable information storage. Popular examples includes the constructions of Wylie et al. (2000); Subbiah and Blough (2005) and Storer et al. (2009).

Wylie et al. (2000) introduced PASIS, a decentralized storage system that offers features like data redundancy, self-maintenance and securing storage. PASIS depends on clients that collect data based on filenames and send it to multiple *storage nodes*. The system is built on (k, n)-secret sharing and permits to rebuild lost shares in a secure way.

Subbiah and Blough (2005) proposed GridSharing as a system that introduces a fault model based on byzantine and leakage-only crashes. It uses a secret sharing scheme based on XOR operations and server replication. The XOR scheme for n participants and a secret file S consists in generating n shares $s_1, s_2, \ldots, s_n$ such that $s_1 \oplus s_2 \oplus \ldots \oplus s_n = S$. Unlike PASIS, GridSharing is built on a *all-or-nothing* secret sharing and thus requires additional replication mechanisms to achieve fault resilience.

Storer et al. (2009) combines the principles of GridSharing and PASIS by adding a pluggable type of independent authorization domains and the possibility of secure migration to new storage nodes. Their system, called POTSHARDS uses two levels of secret sharing (the all-or-nothing XOR scheme and the threshold scheme of Shamir (1979)), approximate pointers and RAID (Redundant Array of Independent Disks) distributed algorithms to recover faulty data. For a survey on RAID, we invite the reader to address the work of Chen et al. (1994).

Other examples of systems that use secret sharing or information dispersal include the work of Rhea et al. (2001) (OceaneStore), Haeberlen et al. (2005) (Glacier), Masinter and Welch (2006), Huhnlein et al. (2009) and Resch and Plank (2011) (AONT-RS). Most of them were experimentally tested or included in commercial systems: OceaneStore prototype was made publicly available as Pond prototype WebSite (2015f); Glacier was used as the storage layer for ePOST serverless email system WebSite (2015g); AONT-RS was implemented in Cleversafe products, which currently provide commercial data storage solutions WebSite (2015b).

We invite the reader to refer to the original papers for more details or to the work of Storer et al. (2006, 2009); Braun et al. (2014) for a more comprehensive list of systems that use secret sharing as a building block for secure storage.

3. ENCRYPTION VS SECRET SHARING

In the literature there exist two distinct approaches to secure storage: (1) by using encryption and (2) by using secret sharing. Subbiah and Blough (2005) and Storer et al. (2009) compare these solutions in the preliminaries of their papers.

The traditional method to build long-term storage systems uses encryption - to preserve data secrecy - and backup techniques - to allow data recovery in case of failure. Fig.2 illustrates this solution in its basic form.

To achieve information confidentiality, data is not stored in clear, but encrypted: each time data is inserted in the database, it is first encrypted (using a cryptographic strong key) and then stored. When an authorized party reads from the database, data is decrypted using the corresponding key. Under the assumption of secure encryption, decryption is computationally infeasible in the absence of the correct key; hence, data remains inaccessible to all unqualified entities. This keeps the information hidden from an adversary even if he gains physical access to the

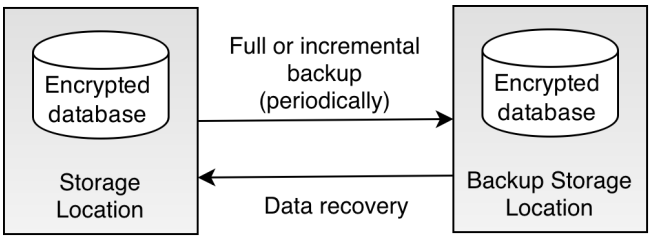

Fig. 2. Database storage architecture using encryption and backup techniques

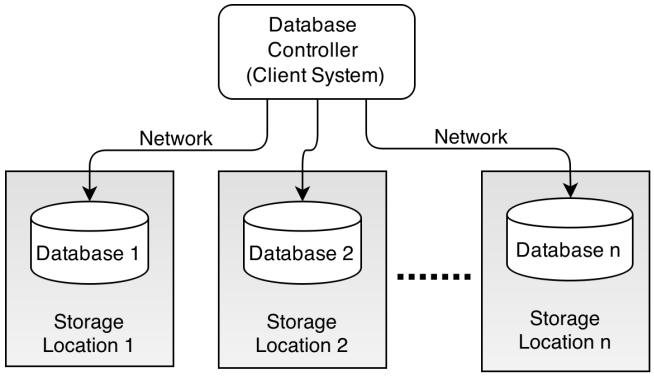

Fig. 3. Database storage architecture using secret sharing techniques

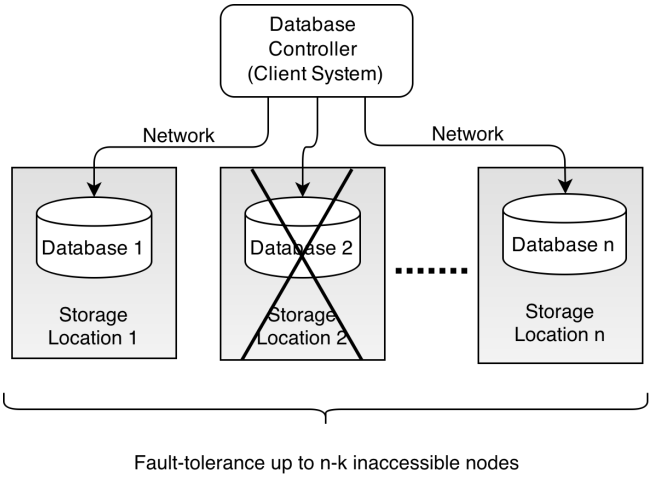

Fig. 4. Fault-tolerance using secret sharing techniques

database. Efficient encryption algorithms exist (e.g. AES), so the solution is fast and practical.

To achieve fault-tolerance, backup or other replication-based techniques are used. Periodically, full and incremental backups replicate data to a distinct physical location. In case of failure, the system performs data recovery from backup.

Secret sharing-based systems represents an alternative solution that provides data secrecy and reliability simultaneously, by splitting the information among multiple storage points, located in different physical locations. Fig.3 illustrates this technique.

To write information, a controller located on the client-side splits the data S into n ($n > 2$) shares and stores each share to a different node. Reconstruction is later possible from at least k out of n ($k \leq n$) shares, where k is chosen accordingly to the system requirements. Hence, when an authorized user wants to read information, the authorized client-side application recovers k or more shares and rebuilds the data. The solution provides perfect secrecy for an adversary that gains access to less than k storage nodes: when the adversary discloses data from less than k storage nodes, he gains no information about the secret data. Since the nodes are located in distinct physical locations and protected by different security mechanisms, it is unlikely for an adversary to access k or more nodes, for k properly chosen.

Fault-tolerance is achieved by construction for $k < n$: the system remains available for up to $n - k$ nodes failures.

Secret sharing-based techniques might be more desirable due to some advantages they provide over encryption-based solutions:

- Encryption is computationally secure, which means that it can only provide confidentiality against a computational limited adversary; an attacker with unbounded computational power can break the secrecy and hence access the data. On the other hand, secret sharing can be theoretically secure, which means that an adversary cannot break its security, regardless the computational power; Shamir (1979) is an example of perfect threshold secret sharing. Theoretical security is important for long-storage systems, since the data must be kept secret for decades or even centuries, during which an assertive adversary can permanently try to break it and eventually succeed.
- Encryption-based techniques must deal with lost or compromised keys; it should also periodically change the keys and hence the key management becomes difficult, in special in case of fine granted access since the number of keys becomes large (note that the keys must be stored securely and reliable). In contrarily, secret sharing permits data reconstruction without any external information, but only from the shares stored in the nodes.
- Verifiable secret sharing achieves data consistency, by attesting the veridity of the shares, and hence of the information that is read from the database (Subbiah and Blough (2005)). Encryption cannot achieve this by default - additional mechanisms must be used.

However, secret sharing is not a perfect solution. First, it combines security and data redundancy, which is sometimes undesirable. Second, it should allow share renewal to prevent information disclosure, since the attacker can gain access to enough storage nodes in time. Third, theoretically secure secret sharing might require large computational overhead; Subbiah and Blough (2005) and Braun et al. (2014) analyze their practical limitations.

Nevertheless, encryption and secret sharing are not disjoint solutions. Constructions that combine encryption and secret sharing exist. An example is CloudSeal, introduced by Xiong et al. (2012), which integrates symmetric encryption and threshold secret sharing along with other mechanisms like proxy-based re-encryption and broadcast revocation to deliver secure storage services in public clouds.

4. THE DATABASE STORAGE SYSTEM

Alouneh et al. (2013) proposed a database partitioning technique with fault-tolerance that they claim to be both efficient and secure. Their proposal is based on the secret sharing scheme of Shamir (1979) and hence maintains the advantages of a (k, n)-threshold secret sharing: the data

is split into n shares such that at least k shares allow reconstruction.

Fig.5 explains shares computation and distribution. Without loss of generality, we assume binary data, after a proper encoding of any electronic document. First, the controller on the client-side splits the binary data into k-bytes blocks, where k is the minimum number of available storage nodes for data recovery. Second, each byte of the block is used as a coefficient of a polynomial in the finite field $GF(256)$. Last, a share i is computed as the value of the polynomial in i, $i = 1, \ldots, n$.

Input: S a binary file from the original database, n the number of the storage nodes, k the threshold required for file recovery

Output: n binary files distributed to multiple storage nodes

Shares Computation: The application on client-side:

- breaks S into k-bytes blocks;
- pads the last block to k-bytes;
- feeds all bytes of a block into the coefficients of a polynomial $f(x) = a_{k-1}x^{k-1} + \ldots + a_1x + a_0$;
- calculates n values $f(1), \ldots, f(n)$;
- repeats the procedure for all blocks of the file S.

Shares Distribution: The application on the client-side securely distributes the values $f(i)$ to the storage node i, $i = 1, ..., n$.

Fig. 5. Alouneh et al. (2013) - Distribution phase

Input: (At least) k binary files stored in distinct nodes

Output: S the binary file of the original database

Shares Computation: The application on the client-side:

- reads the corresponding values $f(i)$ from k distinct nodes;
- performs Lagrange interpolation and recovers the coefficients of the polynomial $f(x) = a_{k-1}x^{k-1} + \ldots + a_1x + a_0$;
- builds a block of the original file as the concatenation of the coefficients;
- repeats the procedure for all blocks of the file S.

Fig. 6. Alouneh et al. (2013) - Reconstruction phase

At reconstruction, each block is rebuilt by polynomial interpolation, from any set A of at least k shares:

$$f(x) = \sum_{i \in A} f(i) \prod_{j \in A, j \neq i} \frac{x - j}{i - j} \quad (1)$$

From the security of Shamir (1979), less than k shares reveal no information about S. Fig.6 explains the reconstruction in detail. Again, all computation is performed in $GF(256)$ using the irreducible polynomial $x^8 + x^4 + x^3 + x + 1$, since all coefficients are bytes, but we omit to specify this every time, since it is obvious from the context.

The main difference from the usual usage of Shamir's scheme in other storage systems is the selection of the polynomial. While in general the polynomial is randomly and uniformly chosen such that its free term represents the data to be shared, Alouneh et al. (2013) use the original data to feed in all the coefficients of the polynomial. This approach considerably diminishes the dimension of the shares, since each share only requires $1/k$ from the original storage (every k bytes of the original file describe the polynomial f and a share i consists in $f(i)$).

However, the system presents an important drawback: it is deterministic (i.e. it always generates the same set of shares for a given input S); hence, contrary to authors claim, the system is insecure. We will motivate this in the next section.

5. VULNERABILITIES

Alouneh et al. (2013) claim that their proposal is both efficient and secure. More precisely, they affirm that data confidentiality is inherited from the original sharing algorithm. In fact, this is not true, because of the way the scheme is used: the polynomial is not randomized, but uniquely determined from the shared data. This means the sharing is deterministic, i.e. a given file always splits into the same shares. Hence, Alouneh et al. (2013) break a basic rule of security: to employ randomness.

The deterministic behavior of the system leads to multiple vulnerabilities and simple attacks. We illustrate next two classes of weaknesses caused by the deterministic nature of the algorithm when the shares are computed in order: file type detection and file content detection. Both vulnerabilities can be exploit by an adversary that gains access to at least one storage node, regardless the threshold k.

Other gaps that exist in the original article might lead to new attacks. For example, Alouneh et al. (2013) does not specify the padding; it is well-known that a bad-chosen padding can lead to system's vulnerabilities.

5.1 File type detection

In computing, a short sequence of bytes placed at the beginning of the file (the file's *signature* or *header*) identifies its format. Table 1 exemplifies the first 4 bytes of some of the most popular file types. More file signatures can be found online at WebSite (2015d).

Without loss of generality, let's consider the case of a *pdf* file. Independently of the content, the first 4 bytes of the file are 25 50 44 46. This means that for $k \leq 4$,

Table 1. File signatures

File Type	First 4 bytes of header			
doc	D0	CF	11	E0
gif	47	49	46	38
pdf	25	50	44	46
png	89	50	4E	47
rar	52	61	72	21
wav	52	49	46	46
zip	50	4B	03	04

Table 2. The shares of the first block ($k = 2$)

File Type	Node 1 ($i = 1$)	Node 2 ($i = 2$)	Node 3 ($i = 3$)	Node 4 ($i = 4$)	Node 5 ($i = 5$)
doc	31	85	154	193	14
gif	14	213	156	120	49
pdf	117	133	213	126	46
png	217	41	121	210	130
rar	51	144	241	205	172
wav	27	192	137	109	36
zip	27	198	141	103	44

the polynomial $f(x)$ that corresponds to the first block is always the same; hence, if the node index i is fixed, the first share always equals a same value $f(i)$, regardless the content of the *pdf* file that is stored.

In consequence, an adversary that gains access to a single storage node can group the sets of shares based on the value of the first share; all sets of shares within the same group correspond to the same file type.

In addition, under the same assumption that the nodes maintain the same index i and the adversary gains knowledge of both i and k, the adversary can determine (with good probability) the file type from the value of the first share. The assumption seems plausible, since k is not secret and i must be known for reconstruction, so it can leak.

To exemplify, an adversary can distinguish (with high probability) between *doc, gif, pdf, png, rar, wav* and *zip* files. Table 2 lists the shares for $k = 2$ and $n = 5$. If the adversary gains access to the first node and reads the share 31, then he learns it corresponds to a *doc* file; similarly, if he gains access to the third node and reads 121, then he learns it corresponds to a *png* file. If the first node is vulnerable and the adversary reads 27 , he knows the file is either *wav* or *zip*; however, he can distinguish between *wav* and *zip* files when disclosing data in any other node i, $i = 2, \ldots, 5$, since the values of the shares that correspond to the two file types are distinct.

5.2 File content type detection

Multiple documents follow a specific template; such examples include contracts, invoices, financial documents and curriculum vitae. Since some of the data within these files remains unchanged and only the filled in information differs, the ratio of common shares can be high. Hence, when the attacker gains access to a database node that contains shares of distinct documents, he can determine the file content type by analogy.

The particular content of file is also easy revealed from a single set of shares. For example, a file containing lots of repeating bytes leads to the existence of duplicate blocks of data and hence duplicate polynomials. In consequence, the shares repeat periodically. Even worst, the existence of all-zeros blocks lead to zero value for all shares.

6. IMPLEMENTATION AND RESULTS

To show the applicability of our observations in practice, we have implemented the proposal of Alouneh et al. (2013) and run some test cases.

For implementation, we used Python 3.0 programming language, under ArchLinux OS. Python is a dynamic programming language available under open source license (WebSite (2015h)). Serializing the data for inter-process communication was done with the Ceralizer package WebSite (2015a), while the shares distribution was plotted using the Matplotlib package (Hunter (2007); WebSite (2015e)).

The original work of Alouneh et al. (2013) skips to mention any padding method; hence, we choose a simple and standardized method for padding: pad the last block with 80 00 00 ... 00 until the size of the block reaches k-bytes. We mention the padding method for completeness only, since padding does not affect our results - the analysis is performed on the header, respectively the beginning of the shared files.

6.1 File type detection

Again, we consider the file types from Table 1.

We now extend the analysis we performed in Subsection 5.1 for $k = 2$ and increase the index i until two shares become equal. Let f_l be the 1-degree polynomial that corresponds to the first block of the file type listed on line l; similarly, let f_c be the 1-degree polynomial that corresponds to the first block of the file type listed on column c. Table 3 lists the maximum i such that $f_l(i) \neq f_c(i)$. Value -1 denotes no collision exists for i, $i = 1, \ldots, 255$ (i.e. $\nexists 1 \leq i \leq 255$ s.t. $f_l(i) = f_c(i)$).

It is immediate that the first diagonal can be ignored, since for $k \leq 4$ all shares are equal. Also, note that Table 3 lists 0 for (*wav*,*zip*) pair, since $f_{wav}(1) = f_{zip}(1) = 27$ as results from Table 2.

Tables 4 and 5 list the results for $k = 3$, respectively $k = 4$. We remark that $k \geq 5$ requires more than four bytes in the header; however, such values for k are more rarely used in practice and we ignore them in our analysis.

Under the exact description of Alouneh et al. (2013) given in Fig.5 a storage node i, $i = 1, \ldots, n$ always receives $f(i)$ as a share (i.e. the shares are computed in order and a node i always receive the value of the polynomial evaluated at i). This means that an adversary that gains access to a single storage node i can distinguish between any two file types with probability 1 if i is less than the value listed in the table. For example, an adversary that gains access to a single storage node in a storage system using $k = 2$ can distinguish between *doc* and *gif* files with probability 1 if $i \leq 169$. Since $n > 169$ is totally impractical, the file

Table 3. Maximum value of i s.t. the shares of the first block are distinct ($k = 2$)

File Type	doc	gif	pdf	png	rar	wav	zip
doc	-	169	194	209	170	206	110
gif	169	-	133	137	75	-1	133
pdf	194	133	-	-1	115	151	133
png	209	137	-1	-	229	147	195
rar	170	75	115	229	-	-1	42
wav	206	-1	151	147	-1	-	0
zip	110	133	133	195	42	0	-

Table 4. Maximum value of i s.t. the shares of the first block are distinct ($k = 3$)

File Type	doc	gif	pdf	png	rar	wav	zip
doc	-	63	-1	-1	-1	-1	-1
gif	63	-	-1	-1	-1	-1	-1
pdf	-1	-1	-	164	-1	119	-1
png	-1	-1	164	-	143	122	129
rar	-1	-1	-1	143	-	143	-1
wav	-1	-1	119	122	143	-	172
zip	-1	-1	-1	129	-1	172	-

Table 5. Maximum value of i s.t. the shares of the first block are distinct ($k = 4$)

File Type	doc	gif	pdf	png	rar	wav	zip
doc	-	-1	38	95	1	95	98
gif	-1	-	-1	-1	167	-1	-1
pdf	38	-1	-	12	11	119	70
png	95	-1	12	-	243	95	148
rar	1	167	11	243	-	-1	94
wav	95	-1	119	95	-1	-	-1
zip	98	-1	70	148	94	-1	-

detection works. As a result of the high values obtained and the multitude of -1, an adversary can distinguish with high probability between the considered file types (except *wav* and *zip* for $k = 2$, as already explained).

The file type detection remains feasible under the weaker assumption that the storage nodes are not indexed in a row, but arbitrary, as long as they maintain the same index for multiple sharings, i.e. the client-side application computes n values $f(i_1), \ldots, f(i_n)$, for distinct $i_1, \ldots, i_n$, but maintains i_j associated to node j. Under this scenario, value -1 corresponds to an attack that succeeds with probability 1 when the adversary gains access to a single storage node. Value -1 is most frequent for $k = 3$, indicating the low security of the system.

6.2 File content type detection

We now consider the scenario of content detection and show how an adversary is able to distinguish if two shares stored in a single node correspond to documents with similar content or not. The attack only assumes that the adversary gains access to a single storage node, regardless the threshold k.

For the experiment, we used three PDF files, available online at WebSite (2015c):

(1) Europass Curriculum Vitae - BG, Bulgaria;

(2) Europass Curriculum Vitae - DK, Denmark;

(3) Europass Mobility - RO, Romania.

The first two files follow the European CV template in Bulgarian and Danish. We note that the language of the template also differs, not only the content - which makes the adversary's guess even harder. The third file has completely different content than the first two, being an Europass Mobility document in Romanian language.

To demonstrate the file content detection vulnerability of Alouneh et al. (2013), we give the three files as input to the $(2, 4)$ storage system. This experiment represents a practical implementation of the scenario described in Fig.1.

Fig.7, Fig.8, Fig.9 and Fig.10 plot the first 2000 shares for each of the three files, as they are stored in the databases in storage nodes 1, 2, 3 and respectively 4. Notice that we assume the index i is fixed, hence for each figure, all polynomials are evaluated at the same i.

An adversary that gains access to a single storage node, learns the view of the shares for the given database. Without loss of generality, suppose the adversary discloses the shares in Fig.7. He can easily notice that the first two files have many common shares and hence have similar content, while they differ from the third file. We emphasize that the adversary must not disclose all the shares to gain this knowledge, since he can conclude this with good probability from only several shares that correspond to the common data, which exist due to the template of the files; in this particular case, the adversary can conclude with

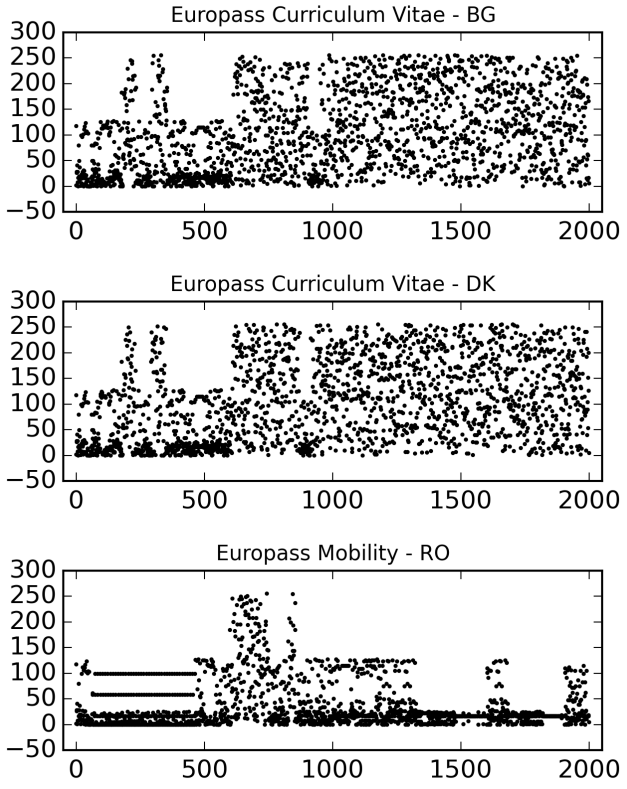

Fig. 7. Database 1: Plot of the shares for the first 2000 blocks

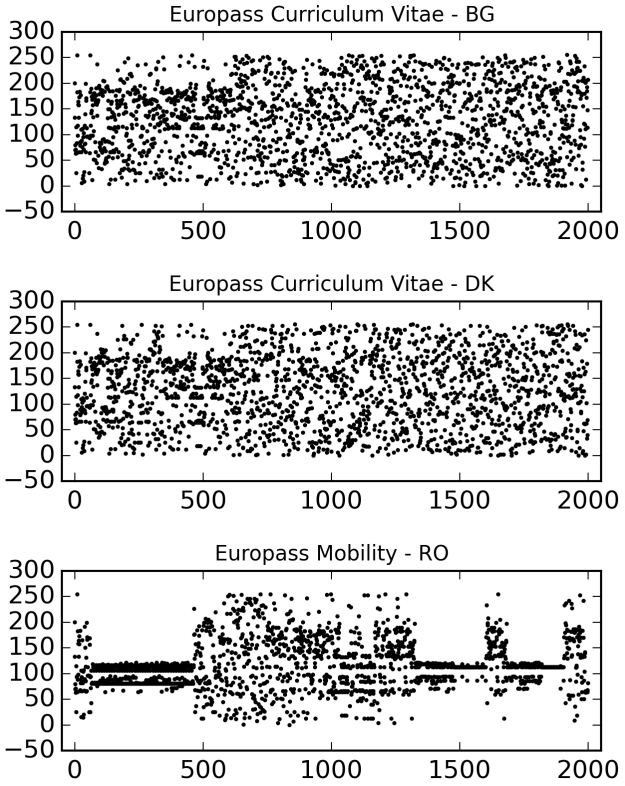

Fig. 8. Database 2: Plot of the shares for the first 2000 blocks

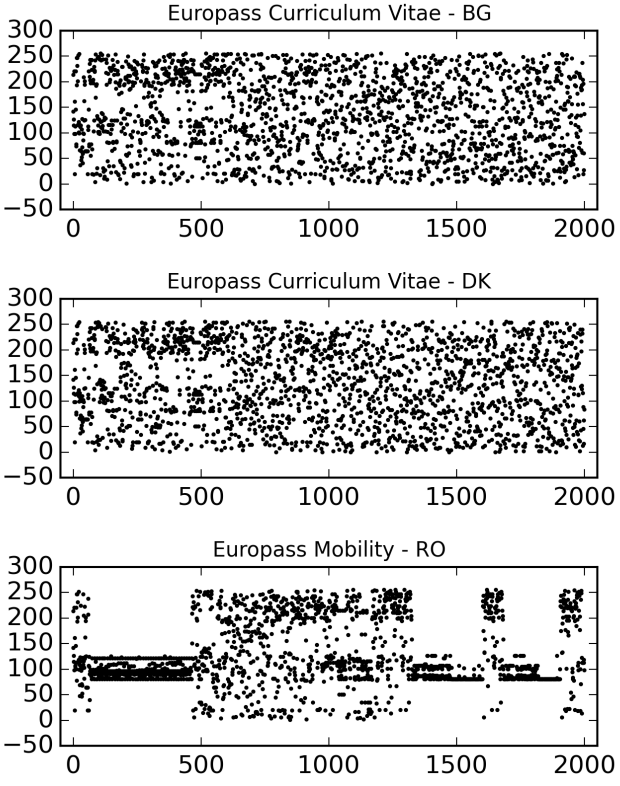

Fig. 9. Database 3: Plot of the shares for the first 2000 blocks

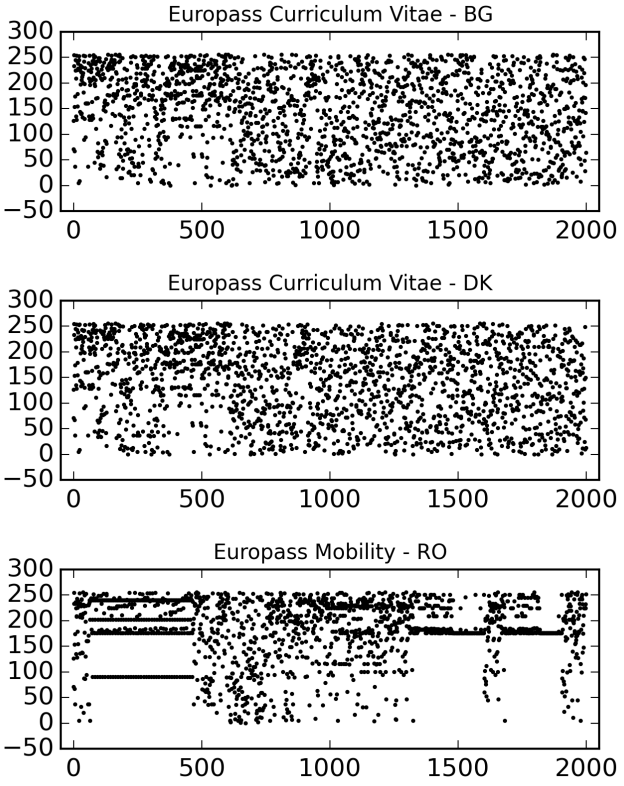

Fig. 10. Database 4: Plot of the shares for the first 2000 blocks

good probability if he only gains access to several shares within the range $0 - 500$.

The difference between the first two and the third subplots visually holds for all four databases, hence the adversary succeeds regardless which node is vulnerable.

A second scenario illustrates the pattern of the shares values for a file that contains periodically repeating information. Let the picture in Fig.11 be the input of the $(2, 4)$ storage system.

Same as before, Fig.12, Fig.13, Fig.14 and Fig.15 plot the shares that are stored in the four distinct located databases. It is visually clear that several values repeat very often; this implies a repeating pattern in the original file. Hence, under the assumption that the adversary gains access to at least one storage node (regardless the threshold of the scheme), he learns if the file contains a repeating pattern.

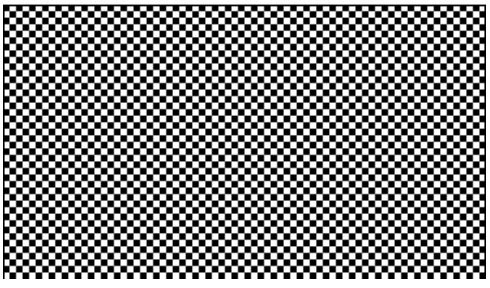

Fig. 11. A pattern-repeating image

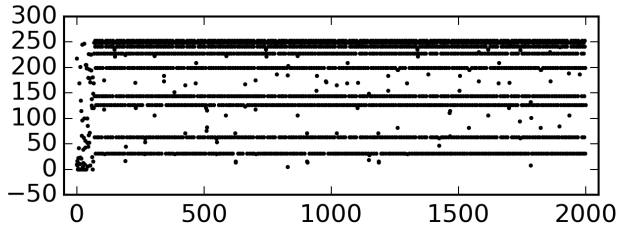

Fig. 12. Database 1: Plot of the shares for the first 2000 blocks

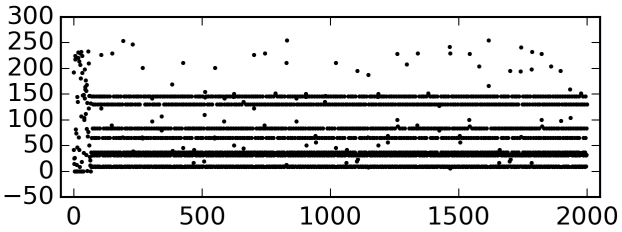

Fig. 13. Database 2: Plot of the shares for the first 2000 blocks

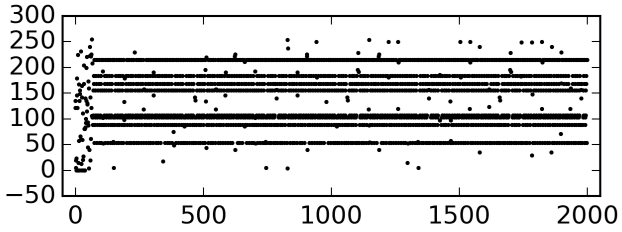

Fig. 14. Database 3: Plot of the shares for the first 2000 blocks

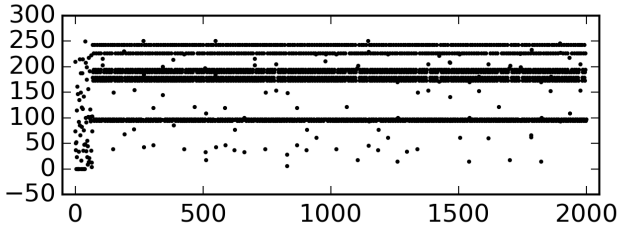

Fig. 15. Database 4: Plot of the shares for the first 2000 blocks

To conclude, we emphasize that all vulnerabilities presented in the current section are caused by the deterministic behavior of the system. In general, a randomized long-storage system based on secret sharing combat such weaknesses, as we will show in the next section.

7. CORRECT USAGE OF SECRET SHARING

Alouneh et al. (2013) adopt threshold secret sharing in an untraditional way to gain efficiency: they use the original data to feed in all the coefficients of the polynomial. But this approach ruins security, as we have shown in the previous section. The traditional usage of Shamir's secret sharing, which consists in feeding only the free coefficient from the input and randomly select the other coefficients, eliminates the deterministic behavior of the system and hence the presented vulnerabilities.

We omit the full description of the algorithms that implement the traditional approach to avoid repetition. The distribution phase is the one in Fig.5, except that a polynomial $f(x)$ is randomly picked for each byte of data: the free coefficient is the data itself, while the others are randomly chosen. The reconstruction phase remains similar to the one in Fig.6, except that it recovers a single byte of data (the free coefficient) from one polynomial interpolation.

It is immediate that under this settings, the system becomes non-deterministic and hence both attacks exploited in the previous section become useless. For completeness, we implemented the randomized algorithms and run the experiments for file content type detection under the same test cases. Note that 2000 blocks in the experiments in Section 6.2 correspond to 4000 bytes of data ($k = 2$) and hence we now plot 4000 shares to consider the same amount of bytes shared from the original file.

Fig.16, Fig.17, Fig.18 and Fig.19 plot the first 4000 shares for each of the three files (Europass Curriculum Vitae -

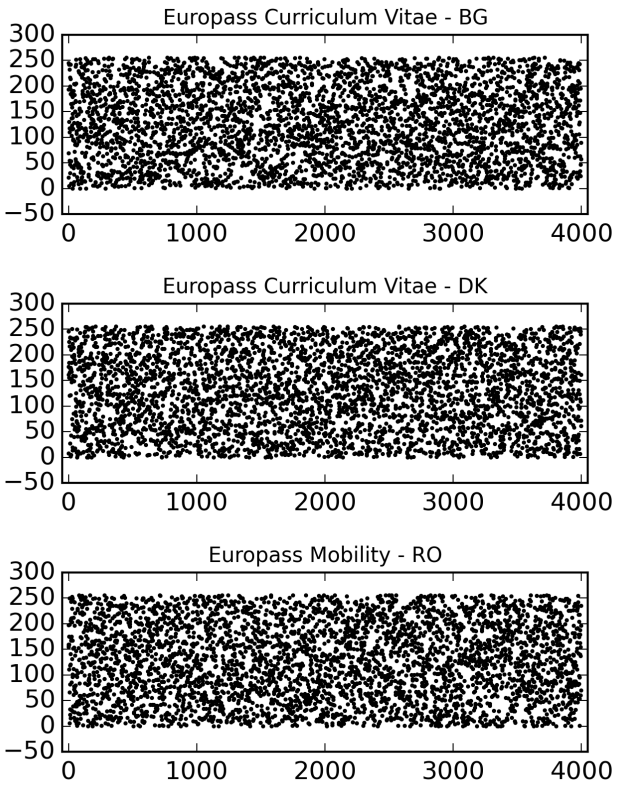

Fig. 16. Database 1: Plot of the shares for the first 4000 bytes

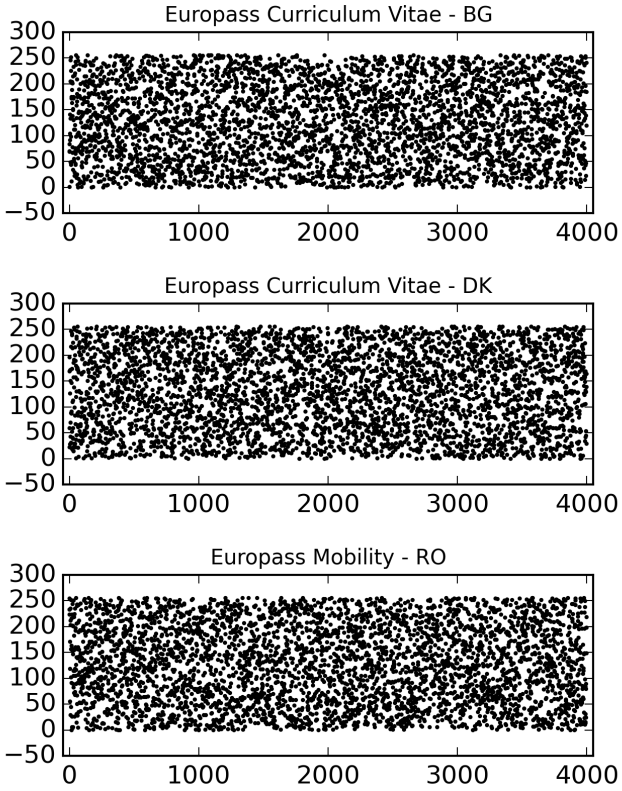

Fig. 17. Database 2: Plot of the shares for the first 4000 bytes

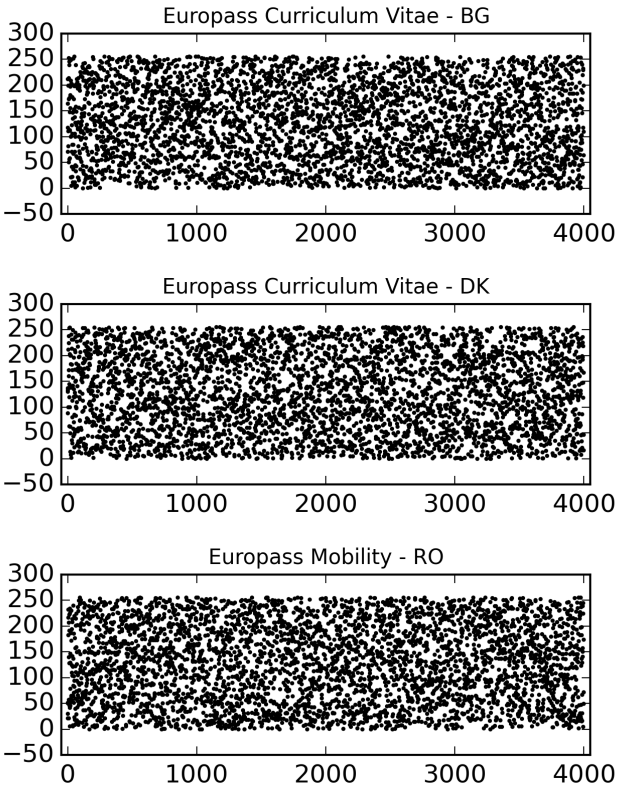

Fig. 18. Database 3: Plot of the shares for the first 4000 bytes

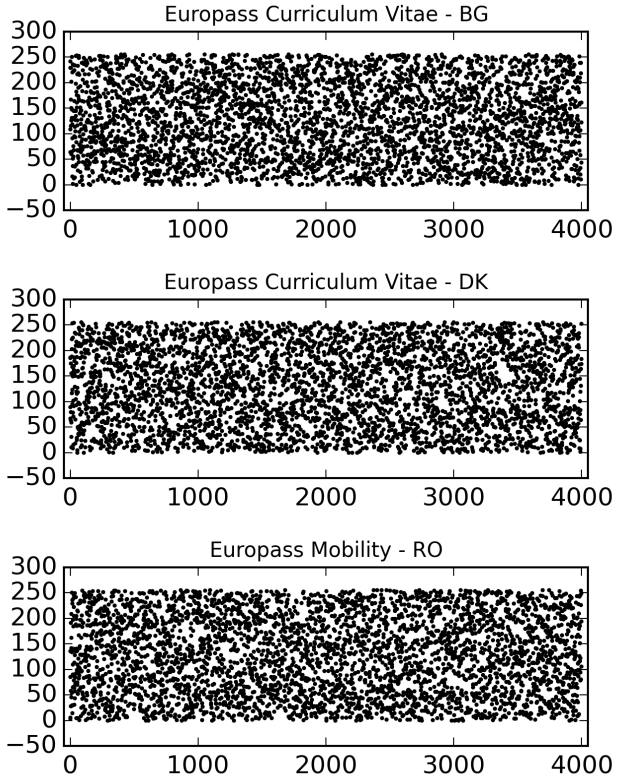

Fig. 19. Database 4: Plot of the shares for the first 4000 bytes

BG, Bulgaria, Europass Curriculum Vitae - DK, Denmark, Europass Mobility - RO, Romania) as they are stored in the databases in storage nodes 1, 2, 3 and respectively 4. In comparison to the system proposed by Alouneh et al. (2013), the graphics reveal no information about the related content of the original files.

The same result holds for the second scenario, when there is no pattern in the shares values obtained by sharing the strong repeating pattern file from Fig.11. Same as before, Fig.20, Fig.21, Fig.22 and Fig.23 plot the shares that are stored in the four distinct located databases. It is visually clear that none of the set of shares indicates a repeating pattern in the original file.

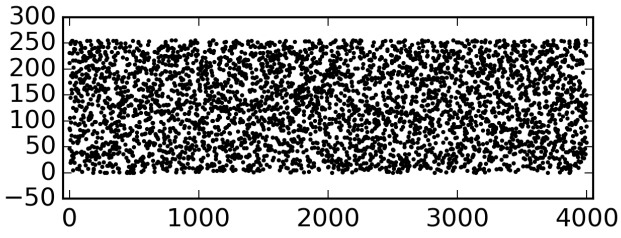

Fig. 20. Database 1: Plot of the shares for the first 4000 bytes

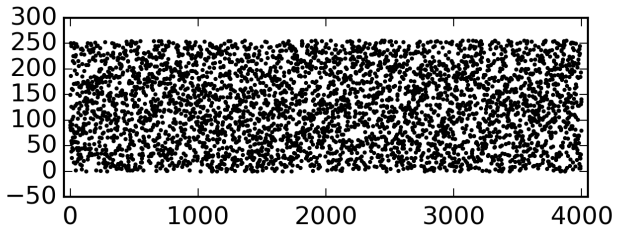

Fig. 21. Database 2: Plot of the shares for the first 4000 bytes

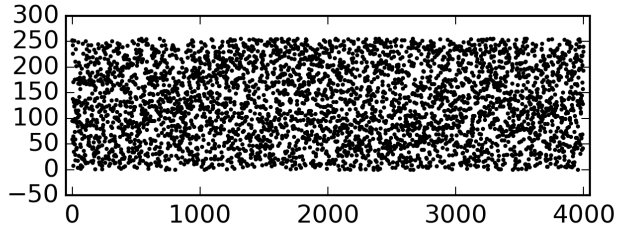

Fig. 22. Database 3: Plot of the shares for the first 4000 bytes

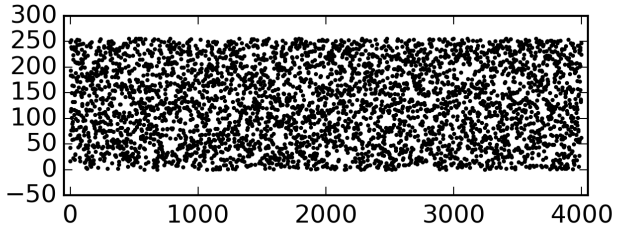

Fig. 23. Database 4: Plot of the shares for the first 4000 bytes

8. CONCLUSION

Alouneh et al. (2013) recently introduced a secret sharing-based technique for long-term storage, which they claim to be both efficient and secure. We proved their assertion is false and reveal some vulnerabilities that are caused by the deterministic behavior of the construction. Both vulnerabilities assume the adversary gains access to at least one storage node, regardless the threshold of the system.

First, we remarked that an adversary can distinguish between common file types or even learn the format or a stored file with non-negligible probability.

Second, we observed that an adversary can distinguish between distinct documents templates, which are stored in the system or he can determine if a file contains repeated patterns.

Finally, to show the applicability of our claims, we implemented the construction of Alouneh et al. (2013) in Python and run some test cases. For file type detection, we considered seven widely spread file types (*doc*, *gif*, *pdf*, *png*, *rar*, *wav*, *zip*), but our analysis can be applied to other types as well; for content detection, we used publicly available filled-in templates and a pattern-repeating black-and-white image. All conducted test cases supported our theoretical observations.

For completeness, we present the traditional way to use secret sharing in storage systems and emphasize that randomization eliminates the presented attacks.

ACKNOWLEDGEMENTS

Ruxandra F. Olimid was supported by the strategic grant POSDRU/159/1.5/S/137750, "Project Doctoral and Post-doctoral programs support for increased competitiveness in Exact Sciences research" cofinanced by the European Social Found within the Sectorial Operational Program Human Resources Development 2007-2013.

REFERENCES

Alouneh, S., Abed, S., Mohd, B.J., and Kharbutli, M. (2013). An efficient backup technique for database systems based on threshold sharing. *JCP*, 8(11), 2980–2989.

Baker, M., Shah, M.A., Rosenthal, D.S.H., Roussopoulos, M., Maniatis, P., Giuli, T.J., and Bungale, P.P. (2006). A fresh look at the reliability of long-term digital storage. In *Proceedings of the 2006 EuroSys Conference*, 221–234.

Blakley, G.R. (1979). Safeguarding cryptographic keys. In *Proceedings of the 1979 AFIPS National Computer Conference*, 313–317.

Braun, J., Buchmann, J.A., Mullan, C., and Wiesmaier, A. (2014). Long term confidentiality: a survey. *Des. Codes Cryptography*, 71(3), 459–478.

Chen, P.M., Lee, E.K., Gibson, G.A., Katz, R.H., and Patterson, D.A. (1994). RAID: high-performance, reliable secondary storage. *ACM Comput. Surv.*, 26(2), 145–185.

Haeberlen, A., Mislove, A., and Druschel, P. (2005). Glacier: Highly durable, decentralized storage despite massive correlated failures. In *Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2*, NSDI'05, 143–158.

Huhnlein, D., Korte, U., Langer, L., and Wiesmaier, A. (2009). A comprehensive reference architecture for trustworthy long-term archiving of sensitive data. In *New Technologies, Mobility and Security (NTMS), 2009 3rd International Conference on*, 1–5.

Hunter, J.D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science and Engineering*, 9(3), 90–95.

Masinter, L. and Welch, M. (2006). A system for long-term document preservation. In *Archiving Conference*, volume 2006, 61–68. Society for Imaging Science and Technology.

Resch, J.K. and Plank, J.S. (2011). AONT-RS: Blending security and performance in dispersed storage systems. In *Proceedings of the 9th USENIX Conference on File and Stroage Technologies*, FAST'11, 14–14.

Rhea, S., Wells, C., Eaton, P., Geels, D., Zhao, B., Weatherspoon, H., and Kubiatowicz, J. (2001). Maintenance-free global data storage. *Internet Computing, IEEE*, 5(5), 40–49.

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

Storer, M.W., Greenan, K.M., and Miller, E.L. (2006). Long-term threats to secure archives. In *Proceedings*

of the 2006 ACM Workshop On Storage Security And Survivability, 9–16.

Storer, M.W., Greenan, K.M., Miller, E.L., and Voruganti, K. (2009). POTSHARDS - A secure, recoverable, long-term archival storage system. *TOS*, 5(2).

Subbiah, A. and Blough, D.M. (2005). An approach for fault tolerant and secure data storage in collaborative work environments. In *Proceedings of the 2005 ACM Workshop On Storage Security And Survivability*, 84–93.

WebSite (2015a). Cerealizer package. Last accessed: July, 2015. `https://pypi.python.org/pypi/Cerealizer`.

WebSite (2015b). Cleversafe. Last accessed: July, 2015. `http://www.cleversafe.com/`.

WebSite (2015c). Europass - download examples. Last accessed: July, 2015. `http://www.europass.cedefop.europa.eu/ro/home`.

WebSite (2015d). File signatures - 512 file signatures. Last accessed: July, 2015. `http://www.filesignatures.net/`.

WebSite (2015e). Matplotlib package. Last accessed: July, 2015. `https://pypi.python.org/pypi/matplotlib`.

WebSite (2015f). Oceanestore. Last accessed: July, 2015. `http://www.oceanstore.org/`.

WebSite (2015g). Oceanestore. Last accessed: July, 2015. `http://www.epostmail.org/`.

WebSite (2015h). Python programming language - official website. Last accessed: May, 2015. `https://www.python.org/`.

Wylie, J.J., Bigrigg, M.W., Strunk, J.D., Ganger, G.R., Kiliççöte, H., and Khosla, P.K. (2000). Survivable information storage systems. *IEEE Computer*, 33(8), 61–68.

Xiong, H., Zhang, X., Yao, D., Wu, X., and Wen, Y. (2012). Towards end-to-end secure content storage and delivery with public cloud. In *Proceedings of the Second ACM Conference on Data and Application Security and Privacy*, CODASPY '12, 257–266.