Convergent Dispersal: Toward Storage-Efficient Security in a Cloud-of-Clouds

Mingqiang Li[1], Chuan Qin[1], Patrick P. C. Lee[1], Jin Li[2]
[1]*The Chinese University of Hong Kong,* [2]*Guangzhou University*
mingqiangli.cn@gmail.com, {cqin, pclee}@cse.cuhk.edu.hk, lijin@gzhu.edu.cn

Abstract

Cloud-of-clouds storage exploits diversity of cloud storage vendors to provide fault tolerance and avoid vendor lock-ins. Its inherent diversity property also enables us to offer keyless data security via dispersal algorithms. However, the keyless security of existing dispersal algorithms relies on the embedded random information, which breaks data deduplication of the dispersed data. To simultaneously enable keyless security and deduplication, we propose a novel dispersal approach called *convergent dispersal*, which replaces original random information with deterministic cryptographic hash information that is derived from the original data but cannot be inferred by attackers without knowing the whole data. We develop two convergent dispersal algorithms, namely CRSSS and CAONT-RS. Our evaluation shows that CRSSS and CAONT-RS provide complementary performance advantages for different parameter settings.

1 Introduction

The advent of cloud computing motivates individuals and enterprises to outsource data storage to the cloud. However, storing all data in a single cloud is susceptible to the single-point-of-failure problem [6] and vendor lock-in [4]. A concept of a *cloud-of-clouds* (also called an *intercloud*) is proposed and studied in recent years [1, 3, 4, 7, 10, 14, 26]. In a cloud-of-clouds, we disperse data, with a certain degree of redundancy, across multiple independent clouds managed by different vendors, such that the stored data can always be available even if a subset of clouds becomes inaccessible.

The cloud-of-clouds model inherently possesses the diversity property, which enables us to offer *keyless* security (without using encryption keys) via dispersal algorithms [8, 15, 22, 24]. By transforming a data object into multiple shares using a dispersal algorithm and dispersing the shares across multiple clouds, the secrecy of the data object can be maintained even if a subset of shares are leaked. Unlike traditional key-based security, keyless security eliminates the need of cloud users to store and manage a large number of keys. However, the keyless security property of existing dispersal algorithms relies on the *random* information embedded in the dispersed data. This randomness prohibits *deduplication* [19] (a technique that exploits content similarity of data to reduce storage space) on the dispersed data.

To provide storage-efficient security, we propose a novel dispersal approach called *convergent dispersal*. The idea is inspired by that of convergent encryption [11], which encrypts data with a cryptographic hash key derived from the data itself rather than a random key. Instead of embedding random information into the dispersed data, convergent dispersal embeds *deterministic* cryptographic hash information that is derived from the original data but cannot be inferred by attackers without knowing the whole data. Thus, the dispersed data preserves content similarity and can be deduplicated.

To this end, we construct two convergent dispersal algorithms, namely *CRSSS* and *CAONT-RS*, both of which augment existing dispersal algorithms with the deduplication property. We show that although the original dispersal algorithms introduce redundancy, both CRSSS and CAONT-RS can effectively reduce storage space via deduplication. We further evaluate the computational throughput of CRSSS and CAONT-RS, and show that the two algorithms provide complementary performance advantages for different parameter settings.

Deployment: Convergent dispersal is applicable to the following deployment scenario. Consider an organization in which users store their backup data in a cloud-of-clouds. The organization may rent a compute server in each cloud for deduplication. To reduce both storage and bandwidth overheads, each user deduplicates its own dispersed data by checking with each compute server for the existence of duplicate data. Cross-user deduplication should be avoided on the user side due to side-channel attacks [13], but instead should be performed by each compute server to achieve additional storage saving. Convergent dispersal enables deduplication on both user and server sides since it preserves content similarity.

2 Background: Dispersal Algorithms

We consider a cloud-of-clouds consisting of n clouds owned by different vendors, as shown in Figure 1. A client disperses its data into the n clouds using a *dispersal algorithm*. The following provides a formal definition

	IDA [20]	**RSSS** [8] with $r \in [0, k-1]$	**SSSS** [24]	**SSMS** [15]	**AONT-RS** [22]
Security Degree	0	r	$k-1$	$k-1$	$k-1$
Storage Blowup	$\frac{n}{k}$	$\frac{n}{k-r}$	n	$\frac{n}{k} + \frac{S_{key}}{S_{secret}} \cdot n$	$\frac{n}{k} + \frac{S_{key}}{S_{secret}} \cdot \frac{n}{k}$
Deduplication	Yes	Yes for $r = 0$; No for $r > 0$	No	No	No

Remark: S_{secret} and S_{key} represent the sizes of a secret and a random key, respectively.

Table 1: Properties of existing dispersal algorithms.

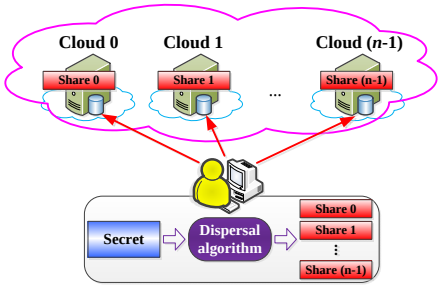

Figure 1: Data dispersal in a cloud-of-clouds.

of general dispersal algorithms:

Definition 1 *An (n, k, r) dispersal algorithm (where $n > k > r \geq 0$) disperses a* secret *into n* shares *such that (1) the secret can be reconstructed from any k shares, and (2) the secret cannot be inferred (even partially) from any r shares.*

A dispersal algorithm achieves both availability and security for the data (called *secret*) to be stored: (1) the secret is available as long as k clouds are accessible (i.e., the algorithm can tolerate the failures of $m = n - k$ clouds); and (2) the secret is secure as long as not more than r clouds are compromised by an attacker. On the other hand, the dispersal algorithm increases the storage space required for storing the data. We define *storage blowup* (denoted by Ω) as the ratio of the resulting data size after dispersal to the original secret size. Note that $\Omega \geq \frac{n}{k}$, since any k out of n clouds contain at least the information of the original secret.

Several dispersal algorithms have been proposed in the literature. We summarize their properties in Table 1. Among these algorithms, the ramp secret sharing scheme (RSSS) [8] with any $r \in [0, k-1]$ is actually the generalization of both Rabin's information dispersal algorithm (IDA) [20] with $r = 0$ and Shamir's secret sharing scheme (SSSS) [24] with $r = k - 1$. In addition, both the secret sharing made short (SSMS) scheme [15] and AONT-RS [22] make improvements over SSSS on storage blowup while keeping $r = k - 1$, but SSMS has slightly higher storage blowup than AONT-RS. Thus, we only choose RSSS and AONT-RS as representatives, and they work as follows.

- RSSS evenly divides the secret into $k - r$ pieces, and generates r *random* pieces of the same size. It then transforms the k pieces into n shares using Rabin's IDA.
- AONT-RS combines Rivest's all-or-nothing transform (AONT) [23] for security and Reed-Solomon (RS) coding [9, 21] for availability. It first transforms the secret into an AONT package with a *random* key, such that an attacker has no knowledge about the AONT package unless the whole package is obtained. The AONT package is then equally divided into k shares, which are finally encoded into n shares that contain the original k shares using a systematic RS code.

Keyless security is realized in real storage systems, such as POTSHARDS [25], DepSky [7], and Cleversafe [22], which build on SSSS, SSMS, and AONT-RS, respectively. Note that the security of existing dispersal algorithms depends on the embedded *random* information (like random pieces in RSSS and a random key in AONT-RS). Due to randomness, distinct secrets with identical content lead to different sets of shares, which would impede data deduplication [19]. To provide storage-efficient security, we need to develop new dispersal algorithms that can produce deduplicable shares while ensuring data security.

3 Convergent Dispersal Algorithms

Inspired by the idea of convergent encryption [11], we propose *convergent dispersal*, which replaces the original random information in existing dispersal algorithms with deterministic hashes generated from the secret. We construct two convergent dispersal algorithms, namely CRSSS and CAONT-RS, which build on RSSS [8] and AONT-RS [22], respectively. We also analyze the deduplication efficiencies for these two algorithms.

3.1 CRSSS

Convergent RSSS (CRSSS) is a convergent dispersal algorithm based on RSSS [8], a generalization of both SSSS [24] and Rabin's IDA [20]. Its basic idea is to replace the r random pieces in RSSS with cryptographic hashes derived from the secret. To ensure security, each hash should be generated using a different hash function. When the size of each random piece is small compared to that of a hash, we can simply fill the r random pieces with hashes generated from the *whole* secret [16]. However, the size of each random piece is often much larger than that of a hash. If we still fill the r random pieces with hashes generated from the whole secret, a large number

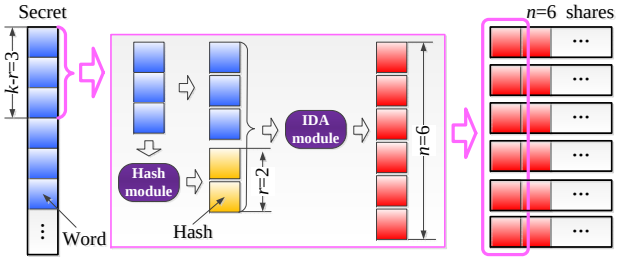

Figure 2: An example of CRSSS with $n = 6$, $k = 5$, and $r = 2$.

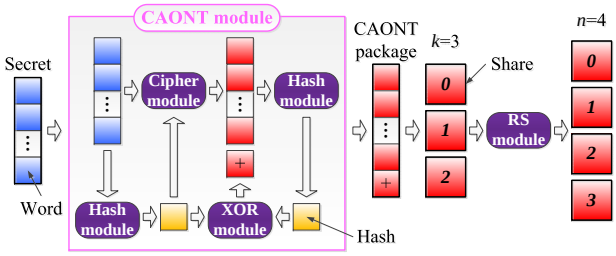

Figure 3: An example of CAONT-RS with $n = 4$ and $k = 3$ (implying $r = 2$).

of hashes of the whole secret have to be generated, which will consume a large amount of computational resource. To reduce computational overhead, we propose to implement CRSSS as follows. Figure 2 shows the idea of the CRSSS algorithm.

Algorithm: Given a secret, we split it into a number of *words* that have the same size as hashes. For example, if we use SHA-256 to generate hashes, the word size is set to be 32 bytes. Each time we process a group of $k - r$ words. We generate r hashes from the $k - r$ words using r different cryptographic hash functions. For example, we can use the following set of r hash functions:

$$h_i = \mathbf{H}(D, i), \text{for } i = 0, 1, \cdots, r-1, \qquad (1)$$

where D represents the data block composed of the $k - r$ secret words, i is an index, and $\mathbf{H}$ is a cryptographic hash function (e.g., SHA-256) that generates the hash h_i from both D and i. The $k - r$ secret words together with the generated r hashes are then transformed into n share words, which will be appended to n different shares (see Figure 2), using Rabin's IDA [20].

Remark: In CRSSS, each of the r hashes cannot be guessed by an attacker unless all the $k - r$ secret words are obtained. Thus, a security degree of r can still be guaranteed. In addition, since there is no random information in CRSSS, distinct secrets with identical content can always be transformed into the same set of shares. Thus, we can perform data deduplication on shares (generated from distinct secrets).

3.2 CAONT-RS

Convergent AONT-RS (CAONT-RS) is a convergent dispersal algorithm based on AONT-RS [22]. Its basic idea is to replace the random key employed in the AONT step of AONT-RS with a cryptographic hash generated from the secret. CAONT-RS comprises two steps: convergent AONT (CAONT) and RS coding, which are described below. Figure 3 shows the idea of CAONT-RS.

Algorithm: The first step is CAONT. Given a secret, we split it into a number of words that have the same size as hashes, as in CRSSS. Suppose that the secret consists of s words. We transform these s secret words into $s + 1$ CAONT words, such that the attacker cannot infer any of the $s + 1$ CAONT words unless all the $s + 1$ CAONT words are obtained. Let $d_0, d_1, \cdots, d_{s-1}$ and $c_0, c_1, \cdots, c_s$ be the s secret words and the $s+1$ CAONT words, respectively. We first generate each of the first s CAONT words by

$$c_i = d_i \oplus \mathbf{E}(h_{key}, i), \text{for } i = 0, 1, \cdots, s-1, \qquad (2)$$

where '$\oplus$' is the XOR operator, h_{key} is the hash key generated from the secret using a cryptographic hash function (like SHA-256), i is an index, and $\mathbf{E}$ is an encryption function (like AES-256) that encrypts the index i with the hash key h_{key}. After generating the first s CAONT words, we generate the last CAONT word by

$$c_s = h_{key} \oplus \mathbf{H}(c_0, c_1, \cdots, c_{s-1}), \qquad (3)$$

where $\mathbf{H}$ is a cryptographic hash function (like SHA-256) that generates a hash from $c_0, c_1, \cdots, c_{s-1}$. The $s + 1$ CAONT words now form a CAONT package.

The second step is RS coding. We evenly divide the CAONT package into k shares. We encode the k shares into n shares using a systematic RS code [9, 21] as in AONT-RS, such that the n shares contain the original k shares.

Remark: In CAONT-RS, the hash key cannot be guessed by an attacker unless the whole CAONT package is obtained. Thus, a security degree of $r = k - 1$ can still be guaranteed. In addition, since there is no random key in the CAONT step, distinct secrets with identical content can always be transformed into the same CAONT package, and hence the same set of shares in the RS coding step. Thus, with CAONT-RS, we can perform deduplication on shares (generated from distinct secrets).

3.3 Analysis on Deduplication Efficiency

We first discuss the configurations of secret and share sizes of CRSSS and CAONT-RS, which may influence their deduplication efficiencies. Suppose that deduplication is performed at the granularity of fixed-size *chunks*. We assume that the share sizes of both CRSSS and CAONT-RS are equal to the chunk size (e.g., 4KB [12]

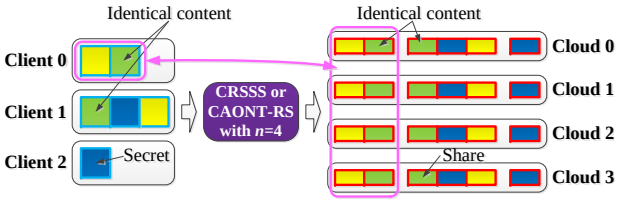

Figure 4: One-to-one mappings between secrets on the client side and shares on the cloud side when $n = 4$.

or 8KB [27]). To produce shares with the chunk size, in CRSSS, we set the secret size to be $k - r$ times the chunk size, while in CAONT-RS, we set the secret size to be $k - 1$ times the chunk size for address alignment. For CAONT-RS, we need to pad zero-filled bytes to the end of the generated CAONT package, so as to form a package that can be equally divided into k shares. Consequently, the storage blowup of CRSSS is still $\Omega = \frac{n}{k-r}$, while that of CAONT-RS becomes $\Omega = \frac{n}{k-1}$.

We now analyze the deduplication efficiencies of both CRSSS and CAONT-RS. In both approaches, there is a one-to-one mapping between each secret and its n shares. Consider an example in Figure 4, in which three clients disperse three data files (composed of a number of secrets) to four clouds. In each of the four clouds, deduplication is performed on the stored files (composed of a number of shares). From this example, we see that the deduplication ratio for encoded shares in each cloud is the same as that for unencoded secrets among all clients (both are $2 : 1$). Suppose that we do not consider the additional storage overhead for the metadata used in data deduplication. Then for both CRSSS and CAONT-RS, we consider a case in which the deduplication ratio for unencoded secrets among all clients is denoted by a single constant $\Lambda : 1$ (e.g., $\Lambda = 20$ for backup workloads [5]). We then define the effective deduplication ratio as:

$$\frac{\Lambda}{\Omega} = \begin{cases} \frac{k-r}{n} \times \Lambda, \text{for CRSSS;} \\ \frac{k-1}{n} \times \Lambda, \text{for CAONT-RS.} \end{cases} \quad (4)$$

We consider the following scenario. Suppose that $\Lambda = 20$, $n = 8$, and $k = 6$. For CRSSS, let $r = 2$, so its effective deduplication ratio is $10 : 1$; for CAONT-RS, its effective deduplication ratio is $12.5 : 1$. We see that both CRSSS and CAONT-RS still have high deduplication efficiency.

In practice, for a given set of data files, the deduplication ratio is determined by different deduplication granularities [27]. Since CRSSS and CAONT-RS have different secret sizes, their actual deduplication ratios can be different. We pose the formal analysis of the deduplication efficiency as future work.

4 Performance Evaluation

We evaluate the computational throughput of our proposed CRSSS and CAONT-RS, which we measure by the total amount of processed secret data divided by the computational time of generating all shares. We implement both CRSSS and CAONT-RS in C. For comparison, we also implement RSSS [8] and AONT-RS [22]. We take SHA-256 and AES-256 as the default hash and encryption functions, respectively, and implement them using OpenSSL Version 1.0.1 [2]. In addition, we implement Rabin's IDA [20] (for both CRSSS and RSSS) and RS coding [9, 21] (for both CAONT-RS and AONT-RS) using Jerasure Version 2.0 [17], which is integrated with GF-complete Version 1.0 [18] for SSE acceleration. We run our tests on an Intel Xeon E5530 server, which has two Quad-Core CPUs at 2.40GHz with SSE4.2 support.

For both CRSSS and CAONT-RS, we set the share size to be 4KB. In each test, we put 1GB of data into a number of secrets of size $4(k - r)$KB for CRSSS or $4(k - 1)$KB for CAONT-RS, and process these secrets in parallel with 8 threads. To ensure fair comparison, both RSSS and AONT-RS use the same secret and share sizes as CRSSS and CAONT-RS, respectively. For both CRSSS and RSSS, we test the ranges of $3 \leq n \leq 12$, $1 \leq m \leq 2$, and $n - m - 3 \leq r \leq n - m - 1$; for both CAONT-RS and AONT-RS, we test the same ranges of n and m, while r is always equal to $n - m - 1$.

Figure 5 presents our test results. We first examine the performance overhead of the convergence in CRSSS and CAONT-RS by comparing them with RSSS and AONT-RS, respectively. The convergence in CRSSS has an overhead of 30.05% on average (in the range from 12.37% to 58.07%), while that in CAONT-RS has an overhead of only 8.03% on average (in the range from 2.19% to 23.13%). The reason is that CRSSS needs to compute more hashes than CAONT-RS.

We next compare CRSSS and CAONT-RS. For a given m, the throughput of CRSSS decreases with n, since CRSSS computes all n shares in the last step via Rabin's IDA; the throughput of CAONT-RS increases with n, since CAONT-RS only needs to compute m shares in the last step via RS coding, which would account for a smaller proportion in the output n shares for a larger n. In the case of $r = n - m - 1$, CRSSS and CAONT-RS provide complementary performance advantages for different values of n: CRSSS has higher throughput when n is small, while CAONT-RS has higher throughput when n is large. Also, for CRSSS, with the decrease of r, its throughput can increase to a very high value (sometimes over 800MB/s); while for CAONT-RS, its throughput is always no more than 400MB/s due to the fixed $r = n - m - 1$. Thus, CRSSS allows a more flexible tradeoff between security and performance than CAONT-RS.

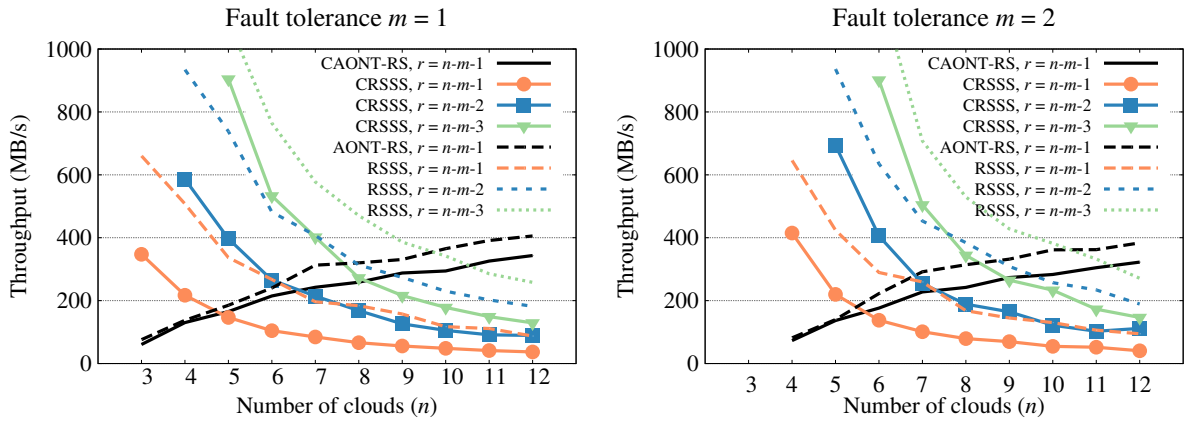

Figure 5: Throughput of various dispersal algorithms.

5 Conclusions

We present convergent dispersal, which supports keyless security and deduplication for cloud-of-clouds storage. Its main idea is to replace random information with deterministic cryptographic hash information derived from original data. We construct two convergent dispersal algorithms, namely CRSSS and CAONT-RS. We analyze their deduplication efficiencies and evaluate their performance via various parameter choices. In future work, we plan to fully implement and evaluate convergent dispersal in a real-life dispersed setting.

Acknowledgments

This work was supported in part by grants from UGC/RGC of Hong Kong (AoE/E-02/08, ECS CUHK419212, and GRF CUHK 413813), National Natural Science Foundation of China (No. 61100224), and the Guangzhou Zhujiang Science and Technology Future Fellow Fund (Grant No. 2012J2200094).

References

[1] IEEE Intercloud Testbed. `http://cloudcomputing.ieee.org/intercloud`, 2014.

[2] OpenSSL Project. `http://www.openssl.org`, 2014.

[3] TClouds. `http://www.tclouds-project.eu/`, 2014.

[4] H. Abu-Libdeh, L. Princehouse, and H. Weatherspoon. RACS: A case for cloud storage diversity. In *Proc. of ACM SoCC*, 2010.

[5] B. Andrews. Straight talk about disk backup with deduplication, 2013. ExaGrid.

[6] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaha. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, Apr 2010.

[7] A. Bessani, M. Correia, B. Quaresma, F. André, and P. Sousa. DepSky: Dependable and secure storage in a cloud-of-clouds. In *Proc. of ACM EuroSys*, Apr 2011.

[8] G. R. Blakley and C. Meadows. Security of ramp schemes. In *Proc. of CRYPTO*, Aug 1984.

[9] J. Blomer, M. Kalfane, R. Karp, M. Karpinski, M. Luby, and D. Zuckerman. An XOR-based erasure-resilient coding scheme. Technical Report TR-95-048, ICSI, UC Berkeley, Aug. 1995.

[10] C. Cachin, R. Haas, and M. Vukolić. Dependable storage in the intercloud. IBM Research Report RZ 3783, May 2010.

[11] J. R. Douceur, A. Adya, W. J. Bolosky, D. Simon, and M. Theimer. Reclaiming space from duplicate files in a serverless distributed file system. In *Proc. of ICDCS*, Jul 2002.

[12] F. Guo and P. Efstathopoulos. Building a high performance deduplication system. In *Proc. of USENIX ATC*, Jun 2011.

[13] D. Harnik, B. Pinkas, and A. Shulman-Peleg. Side channels in cloud services: Deduplication in cloud storage. *IEEE Security & Privacy*, 8(6):40–47, Nov-Dec 2010.

[14] Y. Hu, H. Chen, P. Lee, and Y. Tang. NCCloud: Applying network coding for the storage repair in a cloud-of-clouds. In *Proc. of USENIX FAST*, Feb 2012.

[15] H. Krawczyk. Secret sharing made short. In *Proc. of CRYPTO*, Aug 1994.

[16] J. Li, X. Chen, M. Li, J. Li, P. Lee, and W. Lou. Secure deduplication with efficient and reliable convergent key management. *IEEE Trans. on Parallel and Distributed Systems, PrePrint*, 2013.

[17] J. S. Plank and K. M. Greenan. Jerasure: A library in C facilitating erasure coding for storage applications – version 2.0. Technical Report UT-EECS-14-721, University of Tennessee, Jan 2014.

[18] J. S. Plank, E. L. Miller, K. M. Greenan, B. A. Arnold, J. A. Burnum, A. W. Disney, and A. C. McBride. GF-Complete: A comprehensive open source library for Galois Field arithmetic. version 1.0. Technical Report UT-CS-13-716, University of Tennessee, September 2013.

[19] S. Quinlan and S. Dorward. Venti: a new approach to archival storage. In *Proc. of USENIX FAST*, Jan 2002.

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

[21] I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the SIAM*, 8(2):300–304, 1960.

[22] J. K. Resch and J. S. Plank. AONT-RS: Blending security and performance in dispersed storage systems. In *Proc. of USENIX FAST*, Feb 2011.

[23] R. L. Rivest. All-or-nothing encryption and the package transform. In *Proc. of FSE*, Jan 1997.

[24] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[25] M. W. Storer, K. M. Greenan, E. L. Miller, and K. Voruganti. POTSHARDS—a secure, recoverable, long-term archival storage system. *ACM Trans. on Storage*, 5(2):5, Jun 2009.

[26] M. Vukolić. The Byzantine Empire in the intercloud. *ACM SIGACT News*, 41(3):105–111, Sep 2010.

[27] G. Wallace, F. Douglis, H. Qian, P. Shilane, S. Smaldone, M. Chamness, and W. Hsu. Characteristics of backup workloads in production systems. In *Proc. of USENIX FAST*, Feb 2012.