How to Best Share a Big Secret

Roman Shor* Gala Yadgar* Wentao Huang† Eitan Yaakobi* Jehoshua Bruck‡

*Computer Science Department, Technion, †Snap Inc., ‡California Institute of Technology
{shroman, gala, yaakobi}@cs.technion.ac.il, whuang@snapchat.com, bruck@caltech.edu

ABSTRACT

When sensitive data is stored in the cloud, the only way to ensure its secrecy is by encrypting it before it is uploaded. The emerging multi-cloud model, in which data is stored redundantly in two or more independent clouds, provides an opportunity to protect sensitive data with secret-sharing schemes. Both data-protection approaches are considered computationally expensive, but recent advances reduce their costs considerably: (1) *Hardware acceleration* methods promise to eliminate the computational complexity of encryption, but leave clients with the challenge of securely managing encryption keys. (2) *Secure RAID*, a recently proposed scheme, minimizes the computational overheads of secret sharing, but requires non-negligible storage overhead and random data generation. Each data-protection approach offers different tradeoffs and security guarantees. However, when comparing them, it is difficult to determine which approach will provide the best application-perceived performance, because previous studies were performed before their recent advances were introduced.

To bridge this gap, we present the first end-to-end comparison of state-of-the-art encryption-based and secret sharing data protection approaches. Our evaluation on a local cluster and on a multi-cloud prototype identifies the tipping point at which the bottleneck of data protection shifts from the computational overhead of encoding and random data generation to storage and network bandwidth and global availability.

1 INTRODUCTION

Cloud storage services are ubiquitous, offering high performance and availability, global-scale fault tolerance, file sharing, elasticity, and competitive pricing schemes. Outsourcing data storage and management to a cloud storage provider can be significantly less costly to an organization than maintaining a private data-center with equivalent availability and performance.

However, many businesses and individuals are reluctant to trust an external service provider with their sensitive data; while providers guarantee the durability of the data, they cannot fully guarantee confidentiality in the face of a malicious or compromised employee. Recent reports suggest that the majority of cloud service providers do not specify in their terms of service that data is owned by the customers, and lack security mechanisms to protect it [2, 52]. Furthermore, several incidents of "data leakage" from the cloud have been recently documented [22, 55, 56, 60].

As long as data is stored by one provider, the only way to ensure confidentiality is to encrypt it at the client side, before it is uploaded to the cloud, and decrypt it whenever it is downloaded. This requires generation and maintenance (either locally or remotely) of a large number of encryption keys. Key-based encryption provides *computational security*—it prevents attacks by requiring excessive complexity (and thus, computational power and time). However, because encryption is considered computationally expensive [66, 75], many users still upload their original data to the cloud without further protection [2].

Additional limitations hinder the wider adoption of cloud storage. One is vendor lock-in, where switching from one cloud provider to another (for various business reasons) becomes prohibitively expensive due to the cost of retrieving large amounts of data or developing new application interfaces [54]. Another is outages that a single cloud provider might suffer [34, 40].

An emerging and increasingly popular storage model addresses these limitations; data in a *multi-cloud* [9–11, 15, 47] (also referred to as 'inter-cloud', 'cloud-of-clouds' or 'federated cloud') is stored redundantly in two or more independent clouds. Such redundancy enables users to access or recover their data when one of the clouds is temporarily unavailable, goes out of business, or experiences excessive load. Alternatively, it offers the flexibility of placing more capacity or I/O load on the clouds that currently offer it for the lowest price or highest throughput.

SYSTOR, 2018, Haifa, Israel

ACM ISBN 123-4567-24-567/08/06…$15.00
https://doi.org/10.475/123_4

This new model also presents an opportunity to protect data by *secret sharing*. A secret-sharing scheme is a special encoding which combines the user's original data with redundant random data and ensures that the original data can only be decoded by obtaining all of the encoded pieces. These pieces must be stored on independently secured nodes, as is done in multi-clouds. Secret sharing provides *information-theoretic security*—even an attacker with unlimited computational power has no way of gaining any information about the data that was stored. Thus, information-theoretic security is considered stronger than computational security.

Although they do not require encryption keys, secret-sharing schemes still incur significant storage overhead and non-trivial encoding and decoding complexity, and require generating large amounts of random data. Thus, they are currently used only for long-term archiving of cold data [75], or for remotely storing small amounts of data, like encryption keys [10, 11, 15]. An alternative to explicit storage of encryption keys was proposed in AONT-RS [66]: the keys are hashed with the encrypted data and dispersed on independent storage nodes, achieving significantly higher throughput and lower storage overhead than secret sharing.

Recent technological advances eliminate two major bottlenecks of data protection. One is a new secret sharing scheme, *secure RAID*, that facilitates efficient decoding of partial data, and whose computational overhead is comparable to that of standard erasure coding [29, 33]. Another is hardware-accelerated encryption [23] and its adoption in common cryptographic libraries [3].

These advances present system designers with a new tradeoff. Encryption provides computational security, but requires key generation and management and relies on hardware accelerators for efficient implementation. Secret sharing provides information theoretical security at low complexity but incurs significant storage overhead. Unfortunately, existing evaluation results do not indicate which approach will provide better application-perceived performance, because they are based on studies conducted prior to these advances.

Our goal is to bridge this gap by directly comparing the state of the art of both approaches. We reevaluate the inherent tradeoffs of secure remote storage and present the first comprehensive end-to-end analysis of secret-sharing and encryption-based schemes. Our evaluation addresses all stages of the data path, including random data generation, encoding and encryption overheads, and overall throughput on a local cluster and on geo-distributed remote storage. We implement two secret-sharing schemes and two encryption-based schemes, and measure their performance in a wide range of system parameters, including levels of availability and security, storage devices, and network architectures.

Our main conclusions can be summarized as follows. (1) The low throughput of true random data generation precludes information-theoretical security in real system implementations. (2) Secure RAID completely eliminates the computational bottleneck of secret sharing, and is outperformed only by hardware accelerated encryption. (3) Once storage and network bottlenecks are introduced, secret sharing is outperformed by encryption based techniques due to its additional I/O and transfer overhead. (4) Only encryption and secure-RAID provide efficient access to small random data chunks.

The rest of this paper is organized as follows. Section 2 provides background and Section 3 describes the goals of our analysis. We evaluate computational overheads in Section 4, and end-to-end performance in Section 5. We discuss additional aspects in Section 6 and survey related work in Section 7. Section 8 concludes our work.

2 DATA PROTECTION SCHEMES

Data availability. Fault tolerance in distributed storage systems is provided by replication or by *erasure coding*. An (n, k, r) *erasure code* encodes k *data chunks* into a stripe of n chunks, such that all the data can be reconstructed from any $n - r$ chunks. The encoded chunks are distributed across n different disks or nodes, ensuring that the data remains available even if r arbitrary nodes are unavailable. In a *systematic* erasure code, the original data is stored as is on k nodes and the redundant (parity) information is stored on the remaining $n - k$ nodes. Thus, such a scheme allows direct access to data stored on a healthy node. *Maximum distance separable (MDS)* codes can tolerate the highest number of concurrent node failures given their storage overhead, i.e., $r = n - k$.

The most commonly used erasure code is Reed-Solomon [65], which is both systematic and MDS. Its encoding and decoding entail matrix multiplication over a finite field, traditionally considered a computationally expensive operation. However, efficient implementations of Reed-Solomon are available [62] and used in open-source systems such as Ceph [79] and HDFS [72]. Recent studies show that its encoding and decoding overheads are negligible compared to other overheads in the system [28, 36, 45, 61]. New acceleration libraries, such as Intel's ISA-L [14], utilize specialized processor instructions to further increase encoding and decoding throughput.

Data security. Storage systems must address many aspects of data security, including data integrity, user authentication and access control, and secure communication with clients. These aspects can be successfully guaranteed by any single distributed-storage provider and are orthogonal to our analysis. Mechanisms that address them guarantee that the data stored by users cannot be modified without their consent. However, they do not prevent unauthorized parties from accessing this data, fully or partially.

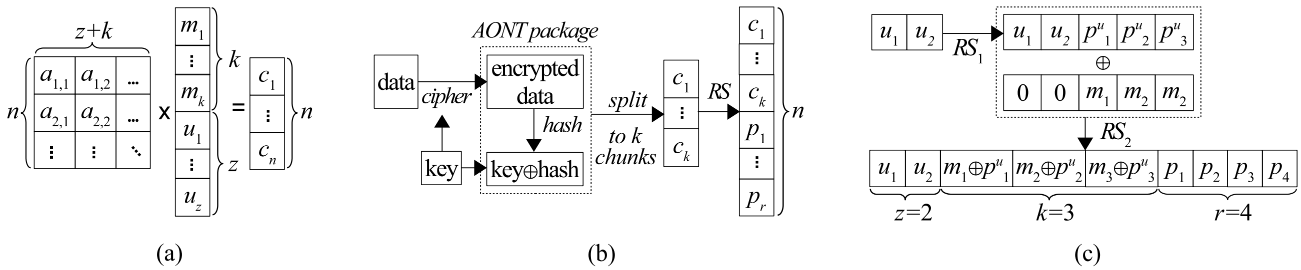

Figure 1: Encoding process of Shamir's secret sharing (a), AONT-RS where $c_1, \ldots, c_k$ are the encrypted data chunks and $p_1 \ldots, p_r$ are Reed-Solomon parities (b), and secure RAID (c).

We note that while unauthorized data modification can be detected by the owner of the data, unauthorized reads can go unnoticed. In this context, *eavesdropping* refers to an unauthorized reader, who might also forward (*leak*) the data or parts of it to an unauthorized third party. *Confidentiality* refers to preventing eavesdroppers from inferring any information about the data. We are interested in the latter in this work.

For data distributed across n nodes, the *confidentiality level* is defined by z, the maximum number of eavesdropping nodes that cannot learn any information about the data, even if they collude. This formal definition inherently assumes that all nodes are independently secured. In other words, when a node is attacked, causing it to behave maliciously, this does not mean the remaining nodes are equally compromised. Thus, the n nodes must be separately managed and owned, like in the multi-cloud model.

Encryption. In *symmetric-key cryptography*, the data is encrypted and decrypted using a small secret *encryption key*. Many distributed storage systems are designed assuming that data has been encrypted at the client prior to being distributed [9, 19, 25, 39, 67]. Thus, generation and maintenance of encryption keys remains the responsibility of the clients. While keys can be generated using a password, these tend to get lost, which results in data loss. Securely storing encryption keys locally at the client prevents access to the data from different end devices, while distributing the keys on several devices introduces additional security issues [48, 58, 75].

Cryptographic encryption introduces significant computational overhead to the data path. The *advanced encryption standard (AES)* [16] is a popular symmetric encryption algorithm, which operates on fixed-length strings (*blocks*) of 128 bits. AES includes implementations (*ciphers*) for key sizes ranging from 128 to 256. Larger encryption keys provide better security, but also incur higher computational overhead. This limitation has recently been addressed by the introduction of a specialized hardware accelerator and a processor instruction set, AES-NI [23].

Secret sharing. Secret sharing is an alternative method for ensuring data confidentiality without requiring maintenance of encryption keys. In an (n, k, r, z) *threshold secret-sharing scheme*, a secret of size k is split between n nodes, such that every subset of z nodes or less cannot deduce any information about the secret, and the data can be recovered if at most r nodes are unavailable [8, 41, 49, 70]. In *Shamir's* generalized secret-sharing scheme[1], also called *ramp* or *packed Shamir* [12], k secrets, $(m_1, \ldots, m_k)$, over a finite field F are shared between n nodes with threshold z as follows. z random elements are chosen from F, $(u_1, \ldots, u_z)$, referred to as *keys* (not to be confused with encryption keys). The secrets and the keys define a polynomial $p(x)$ of degree $z + k - 1$. Evaluating $p(x)$ over n distinct non-zero points $(x_1, \ldots, x_n)$ yields n shares, $c_i = p(x_i)$. Thus the secret can be decoded from any $z + k$ shares, from which the polynomial is reconstructed by interpolation. z shares or less do not reveal any information about the secrets.

The polynomial is typically evaluated via multiplication by a $n \times (z+k)$ matrix, as depicted in Figure 1 (a). Thus, encoding requires $O((z + k)n)$ finite field operations per k secret bytes. Decoding is done by interpolation and incurs $O((z+k)^2)$ finite field operations per byte. Sharing a secret of b bytes requires $\frac{zb}{k}$ bytes of random data. We discuss the challenge of random data generation below. Shamir's generalized scheme can be applied to arbitrary k, r, and z with the minimal achievable storage overhead. However, its main limitation is the need to download $n - r$ non-systematic shares upon every data access.

The added value of confidentiality on top of standard fault tolerance entails significant overhead. It has been shown that the maximal secret size, k, in an (n, k, r, z) threshold secret-sharing scheme is $n - r - z$ [32]. Thus, while the minimal *storage overhead* for tolerating r failures with an erasure code is $\frac{k+r}{k}$ (in MDS codes), the minimal overhead for also tolerating z eavesdropping nodes is $\frac{k+r+z}{k}$.

All-or-Nothing Transform with Reed-Solomon (AONT-RS). AONT-RS [66] was proposed in the context of independently-secure storage nodes, and is designed to avoid the high storage and computational overheads of secret sharing schemes as well as encryption key maintenance. As depicted in Figure 1 (b), it first encrypts the data with a standard symmetric cipher like AES using a random encryption key. It then computes a cryptographic hash of the encrypted data, XORs the hash value with the key, and appends the resulting

[1]Shamir's original scheme required that $k = 1$ [70].

string to the data, creating an *AONT-RS package*. The package is encoded with an (n, k) Reed-Solomon code, and the resulting n chunks are each stored on a different node.

Clients can decrypt any of the systematic chunks as long as they store the encryption key. At the same time, owners who do not store the key locally can recover it by computing the cryptographic hash of all k systematic chunks. This procedure is followed even if the application requires less than k data chunks. An attacker can access the data only by compromising k independent nodes or guessing the encryption key.

Secure RAID. A recently proposed secret-sharing scheme follows an alternative approach for addressing the limitations of Shamir's scheme: rather than relying on encryption, it minimizes the number of finite field operations for encoding and decoding. An (n, k, r, z) *secure-RAID scheme* stores k secrets, $(m_1, \ldots, m_k)$, over a field F. In the first step, z random keys, $(u_1, \ldots, u_z)$, are generated and encoded with an $(n - r, z)$ erasure code and stored systematically on z nodes. In the second step, the k secrets, XORed with the keys and the redundancy generated in the first step, are encoded with an $(n, n - r)$ erasure code and split between the remaining $n - z$ nodes. The security of the scheme is ensured by its combination of erasure codes [29, 33].

Figure 1 (c) shows the encoding in a (9,3,4,2) secure RAID scheme. The two keys, (u_1, u_2), are encoded with a $(5, 2)$ Reed-Solomon code (RS_1) which generates three parities, (p_1^u, p_2^u, p_3^u). These parities are XORed with the secret, (m_1, m_2, m_3), and the result is encoded with a carefully chosen $(9, 5)$ Reed-Solomon code (RS_2) to produce the n shares[2]. Decoding is done by obtaining the keys, encoding them with RS_1, and using the parities to reveal any m_i or all of them. Thus, three shares are required to decode one data share, and any five shares can reveal the entire secret. The data can be recovered from up to four node failures.

This scheme holds several desirable properties. First, its storage overhead is optimal ($k = n - r - z$) as in the generalization of Shamir's scheme. Second, the two encoding steps are comparable in complexity to standard erasure codes. Since the keys are stored systematically and every element of the secret is protected by exactly z keys, the number of finite field operations for encoding is $O(zk + (z + k)r)$. We refer to this property as *near-systematic* encoding. Finally, a random read of a single share of the secret requires accessing only a single encoded share and z keys, and the original share can be decoded with only $O(z)$ finite field operations. This is in contrast to accessing and decoding $n - r$ shares in existing secret-sharing schemes (note that typically, $n - r$ is considerably greater than z).

[2]Secure RAID schemes can be constructed with alternative erasure codes [31]. We focus on the Reed-Solomon-based scheme which does not impose any constraint on k, r, and z.

Random data generation. Key-based encryption and secret-sharing schemes are only as secure as their random data. In *true* random data, the value of one bit does not disclose any information on the value of any other bit. Thus, if the keys are not truly random, an attacker can derive some information about the encoded data.

True random data is generated by measuring a natural source of noise, such as atmospheric or thermal noise, or hardware interrupts [13, 17, 24, 26, 27]. This method produces unpredictable streams of data, but is rate-limited by the external noise source and may require special hardware. Thus, true random data generators are typically orders or magnitude slower than the data protection schemes that rely on them. In addition, most of them cannot be used safely on virtual machines that share hardware [35].

An alternative approach uses a *pseudo-random number generator (PRNG)*. A PRNG is a deterministic algorithm that, given an initial value (*seed*), generates a sequence of uniformly distributed numbers. A *cryptographically secure PRNG (CSPRNG)* generates a random output that is computationally indistinguishable from true random data. Thus, it is considered computationally secure to use CSPRNGs to generate encryption and secret-sharing keys. CSPRNGs are typically implemented with a cryptographic function, whose seed must be generated by a true random generator.

3 CHALLENGES AND GOALS

The schemes described above have been designed with different objectives and tradeoffs between storage and computational overhead, maintenance, and level of security. At the same time, their performance depends on recently introduced acceleration methods for encryption, random data generation, or finite field operations. Thus, previous evaluation results do not provide a clear picture of how these schemes compare in terms of application-perceived read and write throughput. For example, AONT-RS has been shown to outperform Shamir's secret sharing scheme, in a study that preceded both secure RAID and hardware-accelerated encryption [66]. Similarly, the complexity of secure RAID has been shown to be lower than that of Shamir's scheme and encryption, but this theoretical result does not reflect the effects of hardware acceleration on each of these methods. Finally, while secret-sharing schemes rely on large amounts of random data to provide information-theoretical security, we are not aware of any evaluation that includes true random data generation.

To further complicate matters, the benefit of recent schemes and hardware improvements depends on their specific implementation and on the storage system they are applied to. The choice and combination of a random number generator, erasure code, and encryption algorithm can determine which one becomes the bottleneck. Similarly, the system bottleneck

Component	Implementation	Provider	Comments
True RNG	/dev/random RdRand	Linux Intel	Environmental noise as random source, including interrupts and RdRand Thermal noise as random source followed by cryptographic function. Considered here as True RNG due to high seeding rate.
CSPRNG	/dev/urandom AES	Linux OpenSSL (C++)	Based on ChaCha, seeded periodically by the operating system AES 256 in counter mode
PRNG	rand() XOR	<cstdlib> xoroshiro128+	Not secure
Hashing	SHA-1 SHA-256	OpenSSL (C++), Sun (Java) OpenSSL (C++), Sun (Java)	160 bit hash 256 bit hash
Symmetric key encryption	ChaCha AES	OpenSSL (C++), Bouncy Castle (Java) OpenSSL (C++), SunJCE (Java)	Stream cipher, 128 bit keys, used in TLS [18, 42] Block cipher, hardware accelerated using 128, 256 bit keys
Erasure coding	Reed-Solomon (RS)	Jerasure (C++), Backblaze (Java)	Optimized using vectorization with SIMD instructions
Data dispersal	AONT-RS	Our implementation (C++/Java)	AES-128 + SHA-1
Secret sharing	Shamir's Secure RAID	Our implementation (C++/Java) Our implementation (C++/Java)	Uses Jerasure for finite field operations in C++ Based on Reed-Solomon

Table 1: Implementation details of the data protection primitives and schemes used in our evaluation.

may be determined by the speed of the processor, the characteristics of the storage devices, the topology of the network, and the interaction between those components. Multi-cloud environments may further increase the sensitivity of any given scheme to unstable storage and network throughput.

Our goal in this study is to close this gap by mapping the end-to-end costs of the state-of-the art in data protection schemes. To that end, we examine how application read and write throughput are affected by (1) random data generation, (2) hardware acceleration, (3) storage overhead, (4) storage type, and (5) network topology. Our results reveal a different clear winner in each context: in-memory computation, in-house LAN, and multi-cloud.

4 COMPUTATIONAL OVERHEADS

We evaluate the following data protection schemes.

- **Reed-Solomon**, which provides only fault tolerance, is our baseline.
- **Encryption**, which encrypts the data with a key-based symmetric cypher and encodes the result with Reed-Solomon for fault tolerance.
- **AONT-RS**, which hashes the encrypted data, combines the result with the encryption key, and encodes the entire package with Reed-Solomon.
- **Shamir's** secret-sharing scheme, which combines security and fault tolerance in non-systematic encoding.
- **Secure RAID**, which combines security and fault tolerance in two encoding rounds based on Reed-Solomon.

The goal of this section is to evaluate the computational overhead of the presented schemes.

4.1 Methodology

We implemented all the data protection schemes in C++ for the computational performance evaluation and in Java for the distributed objects store described in Section 5. Whenever possible, we based our implementation on existing verified and optimized implementations of standard procedures. The implementation details of the data protection schemes are summarized in Table 1.

	Ciphers			Hash functions (digest)	
	ChaCha	AES-128	AES-256	SHA-1	SHA-256
Encrypt	841.54	4537.93	3379.98	853.73	377.9
Decrypt	846.62	4569.03	3425.19		

Table 2: Throughput (MB/s) of cryptographic functions.

Experimental setup. We performed our evaluations on an 8-core Intel Xeon E5-2630 v3 at 2.40 GHz with 128 GB RAM, running Linux kernel 4.9.0. We first encoded and then decoded 512 4-MB objects (2 GB in total) and measured the single threaded throughput of each data protection scheme. We used random objects generated before the start of the experiment. In each experiment, we varied k (2,4,8,16,32), and r, z (1,2) whenever they were applicable, to reflect a wide range of overheads. We omit $k = \{4, 16\}$ from the results due to lack of space, and include them in the full tech report [71].

We measured the throughput of each scheme in one encode and three decode use-cases.

- **Encode:** n shares were generated from k data chunks. n varied depending on the scheme, either $n = k + r$ for encryption based schemes or $n = k + r + z$ for secret-sharing.
- **Stripe decode:** the k data chunks were generated from k or $k + z$ shares, depending on the scheme.
- **Degraded read:** to emulate one or two lost shares, the k data chunks were generated from the surviving data and parity shares.
- **Random access:** one random data chunk from each stripe was requested and decoded by the corresponding procedure.

4.2 Results

We begin our evaluation with a preliminary comparison of the basic cryptographic primitives that are used by the various data-protection schemes. We measured the encryption and decryption throughput of the ciphers, and the digest throughput of the hash functions. Our results, summarized in Table 2,

True RNG		Secure PRNG		Non-secure PRNG	
/dev/random	RdRand	/dev/urandom	AES	Basic	XOR
1.15	54.82	214.07	3379.98	420.69	798.55

Table 3: Throughput (MB/s) of random data generators.

show that AES achieves a speedup of up to 5x compared to ChaCha, thanks to its hardware acceleration. We use AES-256 in the rest of our evaluation.

Random number generation. We measured the throughput of the RNGs detailed in Table 1. Our results, summarized in Table 3, show that true random data generation is too slow for any practical purpose on a general purpose machine. The AES CSPRNG is the most efficient method, outperforming even the non-secure PRNGs, thanks to its accelerated cipher.

We measured the encoding throughput of Shamir's scheme and secure RAID with random data generated with the different methods to evaluate their overall effect on encoding performance. Figure 2 shows the results for $k = 2, 8, 32$ and $r = z = 2$. They show that the random data generation bottleneck can be eliminated if we are willing to replace information theoretical security with computational security, which can be achieved by hardware accelerated CSPRNG.

To reason about these results, we define the *random rate* as $\frac{z}{k}$, the ratio between the number of random and data bytes in a stripe. Both schemes have the same random rate. Indeed, when $k = 2$ and the random rate is 1, both schemes required 4 MB of random data per 4-MB stripe, and their performance was similar with RdRand, which was the bottleneck. The effect of random data generation decreased with the random rate as k increased. Even with a random rate of 0.0625, RdRand reduced secure RAID's encoding throughput by 3x. Our evaluation of available random number generation techniques leads to our first conclusion, that the low throughput of true random data generation precludes information-theoretical security in real system implementations. In the rest of our evaluation, we used only AES CSPRNG.

Encode/decode. We measured encode, decode and degraded decode throughput of all the schemes. We draw three main conclusions from these results: (1) Secure RAID completely eliminates the computational bottleneck of secret sharing. (2) Hardware accelerated encryption removes the computational overhead and outperforms the other schemes. (3) The performance of AONT-RS is limited by the cryptographic hash.

Figure 3 shows encode (a) and decode (b) throughput of all schemes with $r = z = 2$ and $k = 2, 8, 32$. Reed-Solomon was omitted from the decode experiment because it does not require any decoding. For each encryption based scheme (AES, ChaCha, AONT-RS), the throughput is the same for all k. Hardware accelerated AES performed best among these schemes. The encoding throughput of AES is lower (2160 MB/s), than the AES cipher encryption throughput (3380 MB/s in Table 2) because it also includes Reed-Solomon

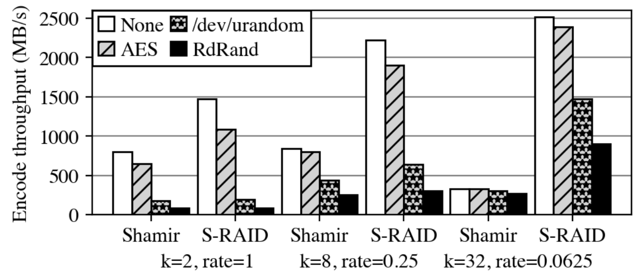

Figure 2: The effect of random data generation on secret-sharing schemes with different random rates and $r = z = 2$.

encoding. AONT-RS had the lowest encoding and decoding throughput, about 650 MB/s. This is due to the low throughput of its hash calculation.

Interestingly, Shamir's encoding and decoding throughput did not increase with k, despite the decreasing random rate. The reason is its non-systematic encoding—the number of operations for encoding grew quadratically with k, and became the bottleneck for $k \geq 4$. Thanks to the near-systematic encoding in secure RAID, its encoding throughput increased with k, as its random rate decreased. Its encoding throughput with $k = 8$ was 1890 MB/s, 55% higher than with $k = 2$, and only 12% lower than hardware accelerated AES. Secure RAID decode throughput is fastest at about 4200 MB/s.

Sensitivity to r and z. We repeated encode and decode measurements with different r and z combinations. The results [71] showed similar trends to encoding and decoding with $z = r = 2$, while efficient schemes were more sensitive to changes in r and z.

Reducing r from 2 to 1 increased the encoding throughput of all schemes with all k values. The increase was higher for the efficient schemes, AES and secure RAID, in which parity generation was responsible for more of the overall overhead. Reducing z from 2 to 1 reduced the random rate and increased encoding and decoding throughput of both secret-sharing schemes. Here, too, the increase was higher in secure RAID which is the more efficient scheme.

Degraded decode. We measured the degraded decode throughput of each scheme when two systematic shares are missing. For encryption based schemes, additional reconstruction overhead affected only AES, whose slowdown was about 36%. Decryption remained the bottleneck of ChaCha and AONT-RS, whose throughput was not affected by the recovery operations. Shamir's scheme was also unaffected, but for a different reason. Due to its non-systematic encoding, every decode had to "recover" k data shares from $n - r$ shares, and the choice of shares did not affect the decoding method. The throughput of degraded decode with secure RAID was roughly half that of regular decode. The throughput increased slightly with an increase in k, as the size of the reconstructed shares decreased [71].

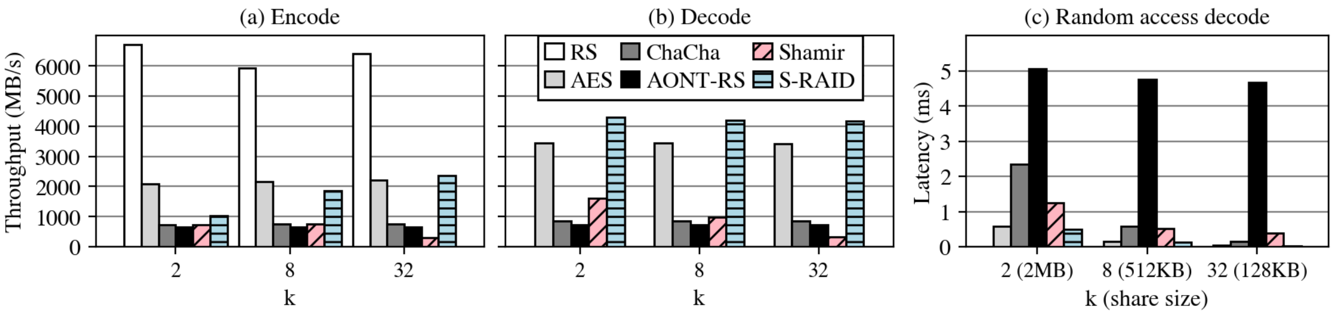

Figure 3: Encoding (a) and decoding (b) throughput and random access decode latency (c) with $r = z = 2$.

	RS	AES	ChaCha	AONT-RS	Shamir	S-RAID
Enc	312.48 (x19)	159.09 (x14)	89.55 (x8)	70.4 (x9)	37.08 (x20)	128.61 (x14)
Dec		664.5 (x5)	121.29 (x7)	111.75 (x6)	65.44 (x15)	297.89 (x14)

Table 4: Measured throughput (MB/s) of main data protection schemes implemented in Java for $k = 8, r = z = 2$ and slowdown (in parentheses) compared to the C++ implementation.

Random access decode. Figure 3 (c) shows the average decoding latency of a single share in a 4 MB object for each scheme. The latency was averaged over decoding of a random chunk from each of the 512 objects. The difference between the data protection approaches is clearly evident, and demonstrates the major limitation of AONT-RS and the major advantage of secure RAID.

The encryption-based schemes had to decode only the requested share, and thus their latency decreased as k increased and the share size decreased. Their measured throughput (not shown) was comparable to that of decoding a full stripe. AONT-RS, on the other hand, had to hash all k shares to obtain the encryption key. This overhead was the bottleneck, preventing the latency from decreasing with share size.

Shamir's scheme had to process almost the entire stripe, $k + z$ shares, to decode a single share. However, the size of its output decreased as k increased, and thus its decoding latency decreased as well. Secure RAID, on the other hand, required only $z + 1$ shares to decode a single share, and is thus faster than all the schemes, and 16–30% faster than AES.

The results of our measurements of encode and decode performance lead to our second main conclusion, that secure RAID completely eliminates the computational bottleneck of secret sharing. Secure RAID is the fastest scheme for decoding, and its encoding throughput is exceeded only by hardware accelerated encryption.

5 END-TO-END EVALUATION

In the previous section, we identified the bottlenecks of the different data protection schemes with respect to their computational overheads. Here, we wish to understand the effect of the various system-level parameters on these bottlenecks, and whether new bottlenecks are introduced. We conducted our evaluation in two different environments. The **LAN** setup consisted of five servers connected by a high speed network. The **multi-cloud** setup consisted of up to 37 virtual servers on Amazon Elastic Compute Cloud (EC2) [1], deployed in multiple geographical regions.

5.1 Methodology

We implemented a distributed object store prototype, which consists of a client that connects to a specified number of servers for transmitting and receiving data shares. We chose Java for our implementation because it provides full and efficient thread management and communication services. Thus, we re-implemented all our data protection schemes in Java.

For consistency, we compared the single threaded encoding and decoding throughput of the data schemes in Java and in C++. Table 4 shows the results for $k = 8$, $r = z = 2$, with the slowdown of the Java implementation compared to that in C++. Although the JNI modules employ optimizations such as vectorization, the achieved increase in throughout is masked by the overhead of data movement between Java and the native modules. To ensure that encoding and decoding are not the bottleneck in our LAN and multicloud setups, the client executes them using a pool of four threads. Our results show that this removes the computational bottleneck for all schemes except Shamir's secret sharing.

Communication was handled by a separate thread for each server and used a secure protocol (TLS v1.2). At the servers, a separate thread managed I/O, to allow I/O and communication to proceed in parallel. Encoding and decoding were executed at the client, which supports one write and four read operations, as follows.

- **Write:** an object of 4MB was encoded into a stripe of n shares with one of our data protection schemes, and transmitted to n servers.
- **Object read:** $n - r$ shares were requested from their servers and decoded.
- **Degraded read:** $n - r$ shares were requested, assuming up to r servers were unavailable. The shares were decoded, possibly with a degraded decode operation.
- **Random read:** one random share was decoded from each

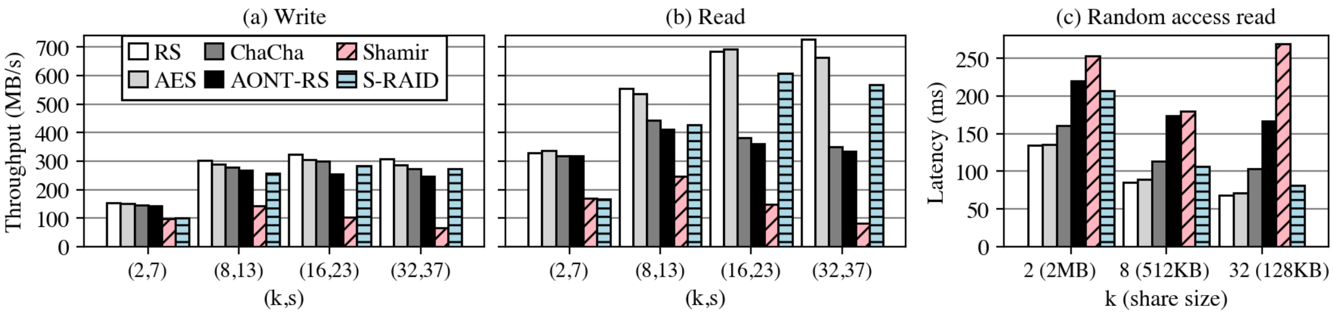

Figure 4: Write (a) and read (b) throughput and random access latency (c) in the LAN setup with $r = z = 2$.

object. The number of servers contacted for this share depended on the data protection scheme.

• **Greedy read:** all n shares were requested from their servers, and decoding began as soon as the first $n - r$ shares were received, possibly as a degraded decode.

We used the same number of servers, s, for each k. We chose s so that $s \geq k+4$, to ensure the n shares were distributed to n different servers. We distributed shares to servers in a round-robin fashion, so that the first chunk of object i was sent to server $i \cdot n(\bmod s)$, and subsequent shares were sent to subsequent servers. For each parameter set and data protection scheme, we wrote a series of 4MB random objects, and then read them with the four read types. The throughput for each operation was measured in a separate experiment and run on a new JVM with clean caches.

LAN Setup. Our local cluster used five machines identical to the one described in Section 4, connected by a 10Gb Ethernet network and equipped with four Dell 960GB SATA SSDs, each. The client ran on a dedicated machine, and each of the remaining machines was used for up to ten virtual servers. Thus, in some of our configurations, some SSDs were serving up to three virtual servers. We ran all combinations of $r = \{1, 2\}$, $z = \{1, 2\}$, and $(k, s) = \{(2, 7), (4, 11), (8, 13), (16, 23), (32, 37)\}$. For each parameter set and data protection scheme, we wrote and read 512 objects, 2 GB in total.

Multi-cloud. We performed the same experiments in the multi-cloud setup, with 256 objects, $r = z = 2$ and $(k, s) = \{(2, 7), (8, 13), (16, 23), (32, 37)\}$. We ran each experiment four times and present the average and standard deviation. We used the same client machine for our multi-cloud setup. We used two instance types for our virtual servers on Amazon's EC2 [4]: `c4.large` has two virtual CPUs, 3.75 GiB of RAM and "moderate network bandwidth". `c4.xlarge` has four virtual CPUs, 7.5 GiB of RAM and "high network bandwidth". We configured our servers with three storage types: *General Purpose SSD* is the default storage provided by Amazon Web Services (AWS), with a baseline throughput of 100 IOPS. *Provisioned IOPS SSD* provides 50 IOPS per 1 GB. We created volumes of 50 GB, with 2500 IOPS per volume. *Throughput Optimized HDD* supports up to 500 MB/s for sequential workloads. Our default setup consisted of `c4.xlarge` machines and general purpose SSDs. We compared the different storage and machine types in a separate experiment, described below.

AWS data centers are divided into *regions*, which correspond to distinct geographical locations and are completely independent. Within a region, isolated data centers are known as *availability zones*. We used separate zones to simulate independent cloud providers. We deployed EC2 instances in 14 different regions and two or three availability zones in each region: Ireland (3), Frankfurt (3), London (3), N. Virginia (3), Ohio (3), N. California (3), Oregon (3), Canada Central (2), Sao Paolo (2), Mumbai (3), Singapore (2), Seoul (2), Tokyo (2), and Sydney (2). Our client machine was located in Israel, which is connected to Europe by fiber optic cables [5].

5.2 Results

The results of our end-to-end evaluation demonstrate how the additional storage overhead of the secret-sharing schemes increases their storage and network bandwidth and limits their performance. They also reinforce the limitation of AONT-RS and Shamir's scheme in small random accesses.

Write/read throughput. Figure 4 shows the write (a) and read (b) throughput of all schemes with $r = z = 2$ and $k = \{2, 8, 16, 32\}$ in the LAN setup. The write and read throughput of Reed-Solomon, AES, and secure RAID increased with k thanks to the reduction in storage overhead and the increased I/O parallelism. Our cluster had 16 SSDs whose utilization increased until the number of servers exceeded the number of devices. Thus, the throughput was maximal with $k = 16$ and slightly lower with $k = 32$, when the overhead of the additional communication threads was considerable.

As the I/O read throughput was higher than write throughput, ChaCha and AONT-RS reached their maximal read throughput with $k = 8$. It did not increase further with k because of their computational overhead. The read and the write throughput of Shamir's scheme did not increase beyond $k = 4$ due to its computational overhead, which was the bottleneck. In secure RAID, the high storage overhead limited its throughput with $k \leq 4$. However, with $k = 16$, the throughput of secure RAID was about 10% lower than that of AES. This

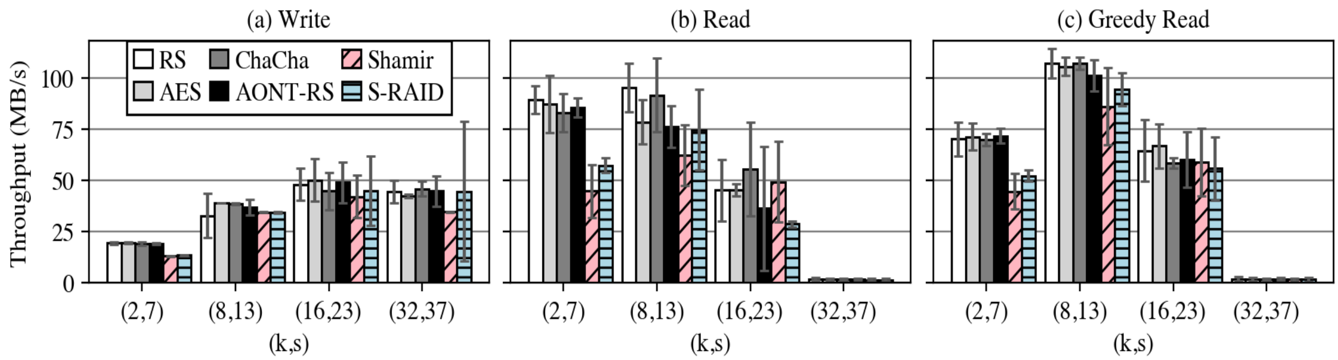

Figure 5: Write (a), read (b) and greedy read (c) throughput in multi-cloud setup, on `c4.xlarge` instances with general purpose SSD storage and $r = z = 2$.

was roughly the difference between the storage overhead of those schemes.

Figure 5 shows the write (a), read (b), and greedy read (c) performance in the multi-cloud setting. The results are averaged over four executions, with error bars marking the standard deviation. The smallest multi-cloud ($s = 7$) was deployed in European regions only. We increased the size of the multi-cloud by deploying instances in additional regions, in order of their observed throughput. As a result, the variability in the throughput provided by different servers increased, increasing the standard deviation of our results.

The write throughput increased with $k = 8$ and $k = 16$, thanks to the increased parallelism, but then decreased with $k = 32$. With $k \geq 8$ the difference between the schemes was no longer noticeable. The read throughput decreased as the number of servers increased, due to the delays induced by high-latency network connections. Our results for the largest multi-cloud ($s = 37$) demonstrate a pathological case; this deployment included two servers each in the Tokyo and Singapore regions, whose observed download throughput was 1.3 Mb/sec and 100Kb/s, respectively. This caused all schemes to achieve extremely low throughput.

The greedy read optimization successfully increased the read throughput with $s = 13$ and $s = 23$, by eliminating the bottleneck of the two slowest servers in each experiment. However, the setup with $s = 37$ included two more slow servers, and the redundancy ($r = 2$) was not high enough to eliminate all of them.

Random access latency. Figure 4 (c) shows the average latency of all schemes when reading one share from a stripe, with $r = z = 2$ and $k = \{2, 8, 32\}$ in the LAN setup. These results reinforce the limitation of AONT-RS and Shamir's scheme with respect to small random accesses.

The latency of Reed-Solomon, AES, ChaCha and secure RAID decreased with k, as the size of the requested share decreased. Secure RAID reads $z + 1 = 3$ shares, because it requires two keys to decode the data share, while the other schemes read only the data. AONT-RS must read and hash the entire object, and thus its latency was higher but decreased slightly with an increase in k, thanks to higher I/O parallelism. Shamir's scheme also reads the entire object. Thus, its latency also decreased as k increased. However, for $k > 8$ its latency increased with k due to the increased decoding complexity.

Storage and server type. We repeated the experiment in the small multi-cloud ($s = 7$) with a different combination of machine and storage type in each run.

The long-distance network bandwidth was the main bottleneck in this experiment, and thus the machine types had little to no effect on the throughput of all operations in all schemes. In contrast, the storage type did affect the throughput of the write and greedy read operations. These operations are less sensitive to the network performance than read, and thus the throughput of all schemes increased with the increase in storage bandwidth provided by the Throughput Optimized HDD, compared to SSD. The results for the Provisioned IOPS SSD, which is optimized for random access, were identical to those of the General Purpose SSD [71].

Our end-to-end evaluation, combining both the LAN and multi-cloud setups, leads to our final two conclusions. First, once storage and network bottlenecks are introduced, secret sharing is outperformed by encryption based techniques due to its additional I/O and transfer overhead. Finally, only encryption and secure RAID provide efficient access to small random data chunks.

6 DISCUSSION

Our evaluation focused on read and write throughput, which are major objectives in storage-system design. However, additional factors affect the applicability and appeal of the different data-protection approaches.

Security level. Our evaluation focused on performance and did not explicitly consider the confidentiality level of each scheme and setup. Namely, the security of secure-RAID and AONT-RS depends on z and k, respectively, while that of encryption is based on its key management scheme.

Full node repair. Recovery of a failed node entails transferring data from the surviving nodes to the *replacement* node in charge of reconstructing the lost data. The replacement node necessarily gains access to more than z shares, which creates a security risk unless the data is encrypted. Several solutions to this problem entail increased storage overhead [6, 59, 68, 69, 77] which, as our results indicate, will likely reduce read and write throughput. In POTSHARDS [75], an additional random mask is transferred with every share, doubling the repair network cost. Methods for minimizing this cost and general reconstruction protocols for any z are studied in [30, 37, 63, 64].

Deduplication. Storage service providers eliminate duplicate data from their systems in order to reduce storage and network costs [20, 21, 73, 81]. Such duplicates cannot be identified when data is encoded before it is uploaded. *Convergent encryption*, in which the encryption key is generated by a cryptographic hash of the data, can successfully alleviate this problem [46, 74]. A similar solution can be applied to secret-sharing schemes [44]. Our results indicate that this will significantly reduce encoding throughput, unless both encryption and hashing are hardware accelerated.

Pricing. Cloud resource pricing depends on the location of the servers, the amount and type of storage attached to them, and the I/O and network bandwidth they use. Therefore, additional storage overhead not only limits the performance of the secret-sharing schemes, but is also more costly for the user. Furthermore, the additional cost of downloading entire stripes during random access or z additional shares in each download may rule out some of the schemes we evaluated.

Our evaluation provides some insight into the effect of several technological trends. As storage-class memory and RAM-based storage [57] gain popularity, the bottlenecks in the data path shift from storage to computation. In such architectures, the bottlenecks we identified in Section 4 may no longer be masked by high storage costs. This may increase the benefit from low computational overhead in schemes like secure RAID, although the additional data transfer they incur may remain the bottleneck. At the same time, hardware acceleration of common complex operations may be applied to additional schemes. Intel's ISA-L acceleration library provides an interface for accelerated Reed-Solomon encoding and cryptographic hashing, which might also be leveraged for random data generation. Such improvements may affect the bottlenecks we identified in Section 4.

7 RELATED WORK

To protect data in a distributed system, several aspects of security must be combined. *Data integrity* refers to ensuring that the data is not modified by anyone other than an authorized user. This is usually obtained by adding cryptographic hashes as signatures to the data before it is stored [19, 25, 39]. Authorized users are *authenticated* by a separate interface, which also verifies user permissions using tokens, access control lists, or other schemes [9, 43, 51, 53]. Communication between the client and the provider's servers, as well as between servers of the same provider, is secured by the network protocols they use [18, 80]. These mechanisms are orthogonal to the scheme used for securely storing the data.

Designing a reliable storage system on a set of untrusted nodes is challenging in several aspects. Early designs that targeted peer-to-peer networks, such as OceanStore [39], Pond [67], and Glacier [25], addressed access control, serialized updates, load balancing, routing, and fault tolerance. They all assume the data has been encrypted prior to being encoded with erasure code and distributed, while maintenance of encryption keys remains the responsibility of the clients. Most multi-cloud architectures follow a similar approach [7, 9, 19]. DepSky [10] and SCFS [11] incorporate encryption into their client, along with a secret-sharing scheme for securely storing the encryption keys. Our results indicate that this is approach is indeed optimal for multi-clouds.

Several studies reduce the storage overhead of secret-sharing schemes by reducing the capacity of individual shares. One approach store only the seed of the randomly generated data, and regenerates it during the decoding process [76]. Our evaluation of the multi-cloud settings indicate that the reduction in storage overhead (and thus, download bandwidth) may justify the increased computational overhead.

Considerable theoretical effort has focused on reducing the computation complexity of Shamir's secret-sharing scheme while still making it information-theoretically secure [29, 31, 41, 49, 50, 78]. Another approach opts for computational security [38]. Our results show that due to the high cost of true random data generation, any implementation of Shamir's and other secret-sharing schemes in a real system will only provide computational security whose strength depends on the strength of the CSPRNG.

8 CONCLUSIONS

We performed the first comprehensive comparison of encryption-based and secret-sharing schemes. We show that information-theoretical security is infeasible in real system implementations, due to the high cost of true random data generation. In terms of encoding and decoding performance, secret sharing with secure RAID is comparable to (and sometimes better than) hardware accelerated encryption.

Our end-to-end evaluation demonstrates how the bottleneck in real implementations shifts from computational complexity to storage throughput (on local storage) and network bandwidth (in cloud deployments). In these settings, encryption outperforms secret sharing thanks to its minimal storage overhead. Thus, our results suggest that encrypting the data and dispersing the keys with an efficient secret sharing scheme is optimal for multi-cloud environments.

REFERENCES

[1] 2006. Amazon EC2. https://aws.amazon.com/ec2/. (2006).

[2] 2017. Netskope Report Reveals Bulk of Cloud Services Still Not GDPR-Ready. https://www.netskope.com/press-releases/netskope-report-reveals-bulk-cloud-services-still-not-gdpr-ready/. (18 Sept. 2017).

[3] 2017. OpenSSL. http://www.openssl.org. (2017).

[4] 2018. Amazon EC2 Instance Types. https://aws.amazon.com/ec2/instance-types/. (2018).

[5] 2018. Submarine Cable Map. https://www.submarinecablemap.com/. (Feb. 2018).

[6] Abhishek Agarwal and Arya Mazumdar. 2015. Security in locally repairable storage. *Manuscript, arXiv:1503.04244* (2015).

[7] Fahad Alsolami and C. Edward Chow. 2013. N-Cloud: improving performance and security in cloud storage. In *IEEE International Conference on High Performance Switching and Routing (HPSR '13)*.

[8] Amos Beimel. 2011. Secret-sharing schemes: a survey. In *International Conference on Coding and Cryptology*.

[9] David Bermbach, Markus Klems, Stefan Tai, and Michael Menzel. 2011. MetaStorage: A Federated Cloud Storage System to Manage Consistency-Latency Tradeoffs. In *IEEE International Conference on Cloud Computing (CLOUD '11)*.

[10] Alysson Bessani, Miguel Correia, Bruno Quaresma, Fernando André, and Paulo Sousa. 2013. DepSky: Dependable and Secure Storage in a Cloud-of-Clouds. *Trans. Storage* 9, 4, Article 12 (Nov. 2013), 33 pages.

[11] Alysson Bessani, Ricardo Mendes, Tiago Oliveira, Nuno Neves, Miguel Correia, Marcelo Pasin, and Paulo Verissimo. 2014. SCFS: A Shared Cloud-backed File System. In *USENIX Annual Technical Conference (ATC '14)*.

[12] George Robert Blakley and Catherine Meadows. 1984. Security of Ramp Schemes. In *Workshop on the Theory and Application of Cryptographic Techniques*.

[13] Marco Bucci, Lucia Germani, Raimondo Luzzi, Alessandro Trifiletti, and Mario Varanonuovo. 2003. A high-speed oscillator-based truly random number source for cryptographic applications on a smart card IC. *IEEE Trans. Comput.* 52, 4 (2003), 403–409.

[14] Dorian Burihabwa, Pascal Felber, Hugues Mercier, and Valerio Schiavoni. 2016. A Performance Evaluation of Erasure Coding Libraries for Cloud-Based Data Stores. In *IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems (DAIS '16)*.

[15] Christian Cachin, Robert Haas, and Marko Vukolic. 2010. *Dependable Storage in the Intercloud*. Technical Report RZ 3783. IBM.

[16] Joan Daemen and Vincent Rijmen. 2013. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer Science & Business Media.

[17] Don Davis, Ross Ihaka, and Philip Fenstermacher. 1994. Cryptographic randomness from air turbulence in disk drives. In *Annual International Cryptology Conference on Advances in Cryptology (CRYPTO '94)*.

[18] Tim Dierks. 2008. The transport layer security (TLS) protocol version 1.2. (2008).

[19] Dan Dobre, Paolo Viotti, and Marko Vukolic. 2014. Hybris: Robust hybrid cloud storage. In *Annual ACM Symposium on Cloud Computing (SOCC '14)*.

[20] Fred Douglis, Abhinav Duggal, Philip Shilane, Tony Wong, Shiqin Yan, and Fabiano Botelho. 2017. The Logic of Physical Garbage Collection in Deduplicating Storage. In *USENIX Conference on File and Storage Technologies (FAST '17)*.

[21] Min Fu, Dan Feng, Yu Hua, Xubin He, Zuoning Chen, Wen Xia, Fangting Huang, and Qing Liu. 2014. Accelerating Restore and Garbage Collection in Deduplication-based Backup Systems via Exploiting Historical Information. In *USENIX Annual Technical Conference (ATC '14)*.

[22] Vindu Goel and Nicole Perlroth. 2016. Yahoo says 1 billion user accounts were hacked. https://www.nytimes.com/2016/12/14/technology/yahoo-hack.html. (14 Dec. 2016).

[23] Shay Gueron. 2010. Intel® Advanced Encryption Standard (AES) New Instructions Set. *Intel Corporation* (2010).

[24] Zvi Gutterman, Benny Pinkas, and Tzachy Reinman. 2006. Analysis of the Linux random number generator. In *IEEE Symposium on Security and Privacy (S&P '06)*.

[25] Andreas Haeberlen, Alan Mislove, and Peter Druschel. 2005. Glacier: Highly durable, decentralized storage despite massive correlated failures. In *USENIX Symposium on Networked Systems Design & Implementation (NSDI '05)*.

[26] Mike Hamburg, Paul Kocher, and Mark E Marson. 2012. Analysis of Intel's Ivy Bridge digital random number generator. Technical Report, Cryptography Research Inc. (Mar. 2012).

[27] W. Timothy Holman, J. Alvin Connelly, and Ahmad B. Dowlatabadi. 1997. An integrated analog/digital random noise source. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications* 44, 6 (1997), 521–528.

[28] Cheng Huang, Huseyin Simitci, Yikang Xu, Aaron Ogus, Brad Calder, Parikshit Gopalan, Jin Li, and Sergey Yekhanin. 2012. Erasure coding in Windows Azure storage. In *USENIX Annual Technical Conference (ATC '12)*.

[29] Wentao Huang and Jehoshua Bruck. 2016. Secure RAID schemes for distributed storage. In *IEEE International Symposium on Information Theory (ISIT '16)*.

[30] Wentao Huang and Jehoshua Bruck. 2017. Generic Secure Repair for Distributed Storage. *CoRR* abs/1706.00500 (2017).

[31] Wentao Huang and Jehoshua Bruck. 2017. Secure RAID schemes from EVENODD and STAR codes. In *IEEE International Symposium on Information Theory (ISIT '17)*.

[32] Wentao Huang, Michael Langberg, Jörg Kliewer, and Jehoshua Bruck. 2015. Communication Efficient Secret Sharing. *CoRR* abs/1505.07515 (2015).

[33] Wentao Huang, Michael Langberg, Joerg Kliewer, and Jehoshua Bruck. 2016. Communication efficient secret sharing. *IEEE Transactions on Information Theory* 62, 12 (2016), 7195–7206.

[34] Jacob Kastrenakes. 2017. Amazon's web servers are down and it's causing trouble across the internet. https://www.theverge.com/2017/2/28/14765042/amazon-s3-outage-causing-trouble. (28 Mar. 2017).

[35] Brendan Kerrigan and Yu Chen. 2012. A study of entropy sources in cloud computers: random number generation on cloud hosts. *Computer Network Security* (2012), 286–298.

[36] Osama Khan, Randal C Burns, James S Plank, William Pierce, and Cheng Huang. 2012. Rethinking Erasure Codes for Cloud File Systems: Minimizing I/O for Recovery and Degraded Reads. In *10th Usenix Conference on File and Storage Technologies (FAST '12)*.

[37] Ankit Singh Rawat Koyluoglu, Onur Ozan and Sriram Vishwanath. 2014. Secure Cooperative Regenerating Codes for Distributed Storage Systems. *IEEE Transactions on Information Theory* 60, 9 (Sept 2014), 5228–5244.

[38] Hugo Krawczyk. 1994. Secret Sharing Made Short. In *Annual International Cryptology Conference on Advances in Cryptology*.

[39] John Kubiatowicz, David Bindel, Yan Chen, Steven Czerwinski, Patrick Eaton, Dennis Geels, Ramakrishan Gummadi, Sean Rhea, Hakim Weatherspoon, Westley Weimer, Chris Wells, and Ben Zhao. 2000. OceanStore: An Architecture for Global-scale Persistent Storage. *SIGPLAN Not.* 35, 11 (Nov. 2000), 190–201.

[40] Rinkesh Kukreja. 2016. The 11 Worst Cloud Outages (Fiascos) of 2016. https://www.stacktunnel.com/worst-cloud-outages-fiascos-2016.html. (2016).

[41] Jun Kurihara, Shinsaku Kiyomoto, Kazuhide Fukushima, and Toshiaki Tanaka. 2008. A new (k, n)-threshold secret sharing scheme and its extension. In *International Conference on Information Security (ISC '08)*.

[42] Adam Langley, Wan-Teh Chang, Nikos Mavrogiannopoulos, Joachim Strombergson, and Simon Josefsson. 2016. *ChaCha20-Poly1305 cipher suites for transport layer security (TLS)*. RFC 7905.

[43] Andrew W. Leung and Ethan L. Miller. 2006. Scalable Security for Large, High Performance Storage Systems. In *ACM Workshop on Storage Security and Survivability (StorageSS '06)*.

[44] Jin Li, Xiaofeng Chen, Mingqiang Li, Jingwei Li, Patrick PC Lee, and Wenjing Lou. 2014. Secure Deduplication with Efficient and Reliable Convergent Key Management. *IEEE Transactions on Parallel and Distributed Systems* 25, 6 (June 2014), 1615–1625.

[45] Mingqiang Li and Patrick P.C. Lee. 2014. Stair codes: A general family of erasure codes for tolerating device and sector failures in practical storage systems. In *USENIX Conference on File and Storage Technologies (FAST '14)*.

[46] Mingqiang Li, Chuan Qin, and Patrick P. C. Lee. 2015. CDStore: Toward Reliable, Secure, and Cost-efficient Cloud Storage via Convergent Dispersal. In *USENIX Conference on Usenix Annual Technical Conference (ATC '15)*.

[47] Mingqiang Li, Chuan Qin, Patrick P. C. Lee, and Jin Li. 2014. Convergent Dispersal: Toward Storage-Efficient Security in a Cloud-of-Clouds. In *USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage '14)*.

[48] Yan Li, Nakul Sanjay Dhotre, Yasuhiro Ohara, Thomas M. Kroeger, Ethan Miller, and Darrell D. E. Long. 2013. Horus: Fine-Grained Encryption-Based Security for Large-Scale Storage. In *USENIX Conference on File and Storage Technologies (FAST '13)*.

[49] Chunli Lv, Xiaoqi Jia, Lijun Tian, Jiwu Jing, and Mingli Sun. 2010. Efficient Ideal Threshold Secret Sharing Schemes Based on EXCLUSIVE-OR Operations. In *International Conference on Network and System Security (NSS '10)*.

[50] Robert J. McEliece and Dilip V. Sarwate. 1981. On Sharing Secrets and Reed-Solomon Codes. *Commun. ACM* 24, 9 (Sept. 1981), 583–584.

[51] Ethan L. Miller, Darrell D. E. Long, William E. Freeman, and Benjamin C. Reed. 2002. Strong Security for Network-attached Storage. In *USENIX Conference on File and Storage Technologies (FAST '02)*.

[52] Mihir Nanavati, Patrick Colp, Bill Aiello, and Andrew Warfield. 2014. Cloud Security: A Gathering Storm. *Commun. ACM* 57, 5 (May 2014), 70–79.

[53] Zhongying Niu, Ke Zhou, Dan Feng, Hong Jiang, Frank Wang, Hua Chai, Wei Xiao, and Chunhua Li. 2007. Implementing and Evaluating Security Controls for an Object-Based Storage System. In *IEEE Conference on Mass Storage Systems and Technologies (MSST '07)*.

[54] Justice Opara-Martins, Reza Sahandi, and Feng Tian. 2016. Critical analysis of vendor lock-in and its impact on cloud computing migration: a business perspective. *Journal of Cloud Computing* 5, 1 (2016), 4.

[55] Dan O'Sullivan. 2017. Cloud Leak: How A Verizon Partner Exposed Millions of Customer Accounts. https://www.upguard.com/breaches/verizon-cloud-leak. (2017).

[56] Dan O'Sullivan. 2017. The RNC Files: Inside the Largest US Voter Data Leak. https://www.upguard.com/breaches/the-rnc-files. (2017).

[57] John Ousterhout, Arjun Gopalan, Ashish Gupta, Ankita Kejriwal, Collin Lee, Behnam Montazeri, Diego Ongaro, Seo Jin Park, Henry Qin, Mendel Rosenblum, Stephen Rumble, Ryan Stutsman, and Stephen Yang. 2015. The RAMCloud Storage System. *ACM Trans. Comput. Syst.* 33, 3, Article 7 (Aug. 2015), 55 pages.

[58] Doyel Pal, Praveenkumar Khethavath, Johnson P. Thomas, and Tingting Chen. 2015. Multilevel Threshold Secret Sharing in Distributed Cloud. In *International Symposium on Security in Computing and Communications (SSCC)*.

[59] Sameer Pawar, Salim El Rouayheb, and Kannan Ramchandran. 2011. Securing Dynamic Distributed Storage Systems Against Eavesdropping and Adversarial Attacks. *IEEE Transactions on Information Theory* 57, 10 (Oct 2011), 6734–6753.

[60] Nicole Perlroth. 2016. Yahoo says hackers stole data on 500 million users in 2014. https://www.nytimes.com/2016/09/23/technology/yahoo-hackers.html. (23 Sept. 2016).

[61] James S. Plank and Mario Blaum. 2014. Sector-Disk (SD) Erasure Codes for Mixed Failure Modes in RAID Systems. *Trans. Storage* 10, 1, Article 4 (Jan. 2014), 4:1–4:17 pages.

[62] James S. Plank, Kevin M. Greenan, and Ethan L. Miller. 2013. Screaming Fast Galois Field Arithmetic Using Intel SIMD Instructions. In *USENIX Conference on File and Storage Technologies (FAST '13)*.

[63] KV Rashmi, Nihar B. Shah, Kannan Ramchandran, and P Vijay Kumar. 2012. Regenerating codes for errors and erasures in distributed storage. In *IEEE International Symposium on Information Theory (ISIT '12)*.

[64] Ankit Singh Rawat, Onur Ozan Koyluoglu, Natalia Silberstein, and Sriram Vishwanath. 2014. Optimal Locally Repairable and Secure Codes for Distributed Storage Systems. *IEEE Transactions on Information Theory* 60, 1 (Jan 2014), 212–236.

[65] Irving S. Reed and Gustave Solomon. 1960. Polynomial codes over certain finite fields. *J. Soc. Indust. Appl. Math.* 8, 2 (1960), 300–304.

[66] Jason K. Resch and James S. Plank. 2011. AONT-RS: blending security and performance in dispersed storage systems. In *USENIX Conference on File and Stroage Technologies (FAST '11)*.

[67] Sean C. Rhea, Patrick R. Eaton, Dennis Geels, Hakim Weatherspoon, Ben Y. Zhao, and John Kubiatowicz. 2003. Pond: the OceanStore prototype. In *USENIX Conference on File and Storage Technologies (FAST '03)*.

[68] Birenjith Sasidharan, P Vijay Kumar, Nihar B. Shah, KV Rashmi, and Kishore Ramachandran. 2014. Optimality of the product-matrix construction for secure MSR regenerating codes. In *International Symposium on Communications, Control and Signal Processing (ISCCSP '14)*.

[69] Nihar B. Shah, KV Rashmi, and P Vijay Kumar. 2011. Information-Theoretically Secure Regenerating Codes for Distributed Storage. In *IEEE Global Telecommunications Conference (GLOBECOM '11)*.

[70] Adi Shamir. 1979. How to Share a Secret. *Commun. ACM* 22, 11 (Nov. 1979), 612–613.

[71] Roman Shor. 2018. *Efficiently Combining Confidentiality and Availability in Distributed Storage Systems*. Master's thesis. Technion, Israel Institute of Technology.

[72] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. 2010. The Hadoop Distributed File System. In *IEEE Symposium on Mass Storage Systems and Technologies (MSST '10)*.

[73] Kiran Srinivasan, Tim Bisson, Garth Goodson, and Kaladhar Voruganti. 2012. iDedup: Latency-aware, inline data deduplication for primary storage. In *USENIX Conference on File and Storage Technologies (FAST '12)*.

[74] Mark W. Storer, Kevin Greenan, Darrell D.E. Long, and Ethan L. Miller. 2008. Secure Data Deduplication. In *ACM International Workshop on Storage Security and Survivability (StorageSS '08)*.

[75] Mark W. Storer, Kevin M. Greenan, Ethan L. Miller, and Kaladhar Voruganti. 2009. POTSHARDS - a secure, recoverable, long-term archival storage system. *ACM Transactions on Storage* 5, 2 (2009), 1–35.

[76] Satoshi Takahashi and Keiichi Iwamura. 2013. Secret Sharing Scheme Suitable for Cloud Computing. In *IEEE International Conference on Advanced Information Networking and Applications (AINA '13)*.

[77] Ravi Tandon, SaiDhiraj Amuru, Thomas Charles Clancy, and Richard Michael Buehrer. 2016. Toward Optimal Secure Distributed Storage Systems With Exact Repair. *IEEE Transactions on Information*

Theory 62, 6 (2016), 3477–3492.

[78] Yongge Wang. 2015. Privacy-Preserving Data Storage in Cloud Using Array BP-XOR Codes. *IEEE Transactions on Cloud Computing* 3, 4 (Oct 2015), 425–435.

[79] Sage A. Weil, Scott A. Brandt, Ethan L. Miller, Darrell D. E. Long, and Carlos Maltzahn. 2006. Ceph: A Scalable, High-performance Distributed File System. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI '06)*.

[80] Aaron D. Wyner. 1975. The wire-tap channel. *The Bell System Technical Journal* 54, 8 (Oct 1975), 1355–1387.

[81] Benjamin Zhu, Kai Li, and Hugo Patterson. 2008. Avoiding the Disk Bottleneck in the Data Domain Deduplication File System. In *USENIX Conference on File and Storage Technologies (FAST '08)*.