Efficiently Combining Confidentiality and Availability in Distributed Storage Systems

Roman Shor

Efficiently Combining Confidentiality and Availability in Distributed Storage Systems

Research Thesis

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science

Roman Shor

Submitted to the Senate
of the Technion — Israel Institute of Technology
Adar 5778 Haifa March 2018

This research was carried out under the supervision of Dr. Gala Yadgar, Prof. Eitan Yaakobi and Prof. Assaf Schuster, in the Faculty of Computer Science.

Acknowledgements

I would like to thank my advisors, especially Gala Yadgar, for their support and all the time invested in me and in this research. I learned a lot their tremendous research experience. It was a pleasure and an privilege working with them and learning from them. I would also like to thank my wife and my parents for supporting me throughout this challenging period.

The generous financial help of the Technion is gratefully acknowledged.

Contents

List of Figures

Abstract

When sensitive data is stored in the cloud, the only way to ensure its secrecy is by encrypting it before it is uploaded. Recently introduced hardware acceleration methods promise to eliminate the computational complexity of encryption, but leave clients with the challenge of securely managing encryption keys. At the same time, 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. Secure RAID, a recently proposed scheme, minimizes the computational overheads of secret sharing, but requires non-negligible storage overhead and random data generation. These recent advances introduce new opportunities to reduce data protection costs considerably. However, previous studies were performed before they were introduced, and thus do not indicate which approach will provide the best application-perceived performance.

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. In this study we implement two secret-sharing schemes and two encryption-based schemes, and measure their performance in a wide range of system parameters. We address all stages of the data path, including random data generation, encoding and encryption overheads, and overall throughput. 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.

Abbreviations and Notations

n	—	Code length
k	—	Data blocks
r	—	Redundancy blocks
z	—	Confidentiality level, the maximum number of eavesdropping nodes that cannot gain any information on the secret
s	—	Number of servers in distributed object store
Random rate	—	$\frac{z}{k}$, the ratio between the amount of random and data bytes in a secret-sharing stripe
Cloud storage	—	Storage model in which data is stored in logical pools accessible via the Internet, while physical storage spans multiple servers in remote data center
Multi-cloud	—	Storage model where data is stored redundantly in two or more independent clouds
MDS	—	Maximum distance seperable, a MDS code with r parity blocks can recover from r node failure
Systematic	—	Code where k data blocks are kept without change
HDD	—	Hard disk drive, a magnetic storage device
SSD	—	Solid-state drive, a flash based storage device
AES	—	Advanced encryption standard, symmetric block cipher
ChaCha	—	Symmetric stream cipher
AONT	—	All or nothing transform
RS	—	Reed-Solomon erasure code
GF	—	Galois finite field
SIMD	—	Single instruction multiple data, processor instructions for vectorization acceleration
RNG	—	Random number generator
PRNG	—	Pseudo random number generator
CSPRNG	—	Cryptographically secure pseudo random number generator

Chapter 1

Introduction

1.1 Cloud Storage

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 [5, 63] 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. Furthermore, several incidents of "data leakage" from the cloud have been recently documented [30, 66, 67, 71].

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 [65]. Another is outages that a single cloud provider might suffer [44, 51].

An emerging and increasingly popular storage model addresses these limitations; data in a *multi-cloud* [12, 14, 15, 22, 58] (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.

1.2 Data Confidentiality

When 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 [78, 87], many users still upload their original data to the cloud without further protection [5].

The multi-cloud model 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.

Secret-sharing schemes do not require encryption-keys, but they incur significant storage overhead, 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 [87], or for remotely storing small amounts of data, like encryption keys [14, 15, 22].

An alternative to secret-sharing schemes was proposed in AONT-RS [78]. This scheme is based on encryption, but instead of explicit encryption keys storage, the keys are hashed with the encrypted data and dispersed on independent storage nodes. This allows AONT-RS to achieve 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 [39, 43]. Another is hardware-accelerated encryption [31] and its adoption in common cryptographic libraries [6].

1.3 Our Contribution

Recent technological advances present system designers with a new trade-off. 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 trade-offs 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.

Chapter 2

Data Protection Schemes

2.1 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 and requires less encoding operations, than non-systematic codes.

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 [77], which is both systematic and MDS. Its encoding and decoding entail matrix multiplication over a finite field. Figure 2.1 depicts (n, k) Reed-Solomon encoding using an $n \times k$ systematic generator matrix. This matrix is multiplied by the $m_1, \ldots, m_k$ data vector to yield data + parity

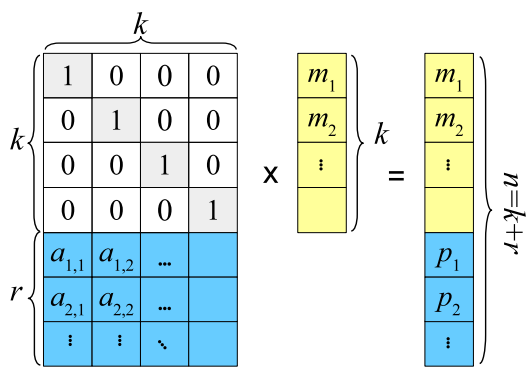

Figure 2.1: Systematic encoding of (n, k) Reed-Solomon erasure code. $m_1, \ldots, m_k$ are the data elements, and $p_1 \ldots, p_r$ are parity elements.

vector. Traditionally, finite field operations are considered computationally expensive. However, efficient implementations of Reed-Solomon are available [73] and used in open-source systems such as Ceph [92] and HDFS [84]. Recent studies show that its encoding and decoding overheads are negligible compared to other overheads in the system [36, 46, 56, 72]. New acceleration libraries, such as Intel's ISA-L [21], utilize specialized processor instructions to further increase encoding and decoding throughput.

Array codes such as EVENODD [18] and RDP [23] employ simple XOR operations during encoding and decoding in order to avoid computationally expensive finite field operations. These codes improve computational complexity (compared to scalar codes, such as Reed-Solomon) by requiring only binary (XOR) operations, at the cost of restricting the choice of encoding parameters and data layout. Another popular erasure code is LRC (Local Reconstruction Code), which is used in Windows Azure [37]. This code adds local parity blocks, so that one node can be recovered by accessing only a small subset of the surviving blocks. Unlike Reed-Solomon, LRC is non-MDS, it introduces a trade-off between storage efficiency and repair cost. We chose to focus on Reed-Solomon erasure code in this study, as it is a widely used, with a variety of efficient open source implementations and acceleration libraries, and can be constructed for any n, k unlike array codes.

2.2 Data Confidentiality

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.

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 gain access to any part of 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.

2.2.1 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 [12, 27, 33, 50, 79]. 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 [59, 69, 87].

Cryptographic encryption introduces significant computational overhead to the data path. The *advanced encryption standard (AES)* [24] 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 [31].

2.2.2 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 [11, 52, 60, 83].

The most prevalent secret-sharing scheme is *Shamir's* [83]. A secret m over a finite field F is 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 secret and the keys define a polynomial $p(x) = m + u_1 x + \cdots + u_z x^z$. Evaluating $p(x)$ over n distinct non-zero points $(x_1, \ldots, x_n)$, yields n shares, $c_i = p(x_i)$. The secret can be decoded from any $z + 1$ shares, from which the polynomial is reconstructed by interpolation. The secret is $p(0)$. The polynomial cannot be reconstructed by less than $z + 1$ shares, so z shares or less do not reveal any information on the secret. Thus, in this scheme, $k = 1$ and $r = n - (z + 1)$.

The polynomial is typically evaluated via multiplication by a $n \times (z + 1)$ matrix, as depicted in Figure 2.2 (a). Overall, encoding the secret requires $O(zn)$ finite field operations per byte. Decoding is typically done by interpolation, incurring $O(z^2)$ finite field operations per byte. Encoding a secret of b bytes also requires zb bytes of random data for the keys. We discuss the challenge of random data generation below.

A generalization of Shamir's secret-sharing scheme, called *ramp* or *packed Shamir* [16], allows r to be independently specified in addition to n and z. Thus, while at least $z + 1$ nodes are required to cooperate in order to gain *any* information on the secret, $n - r$

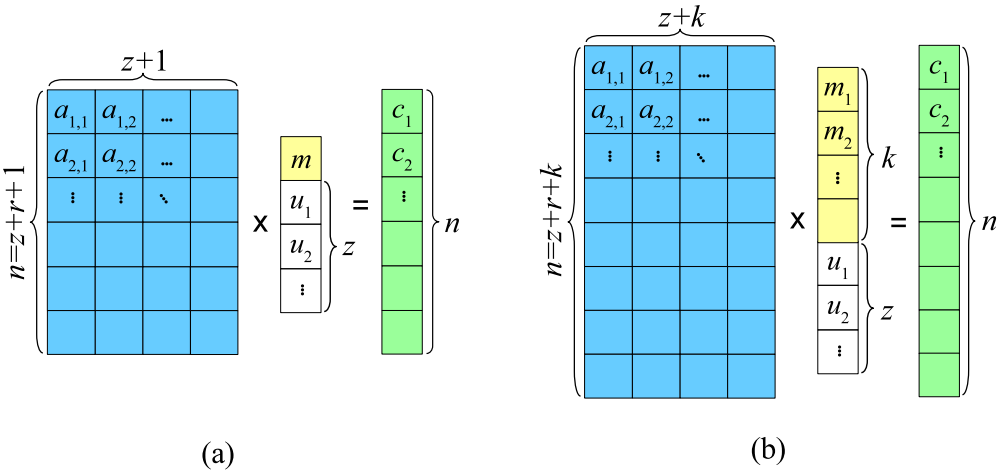

Figure 2.2: Encoding of Shamir's secret-sharing scheme (a) and generalized Shamir's secret-sharing scheme (b). $u_1, \ldots, u_z$ are the random key elements, $m_1, \ldots, m_k$ are the data elements.

nodes are required in order to *fully* recover the secret.

Encoding is similar to Shamir's but is applied to k secrets, $(m_1, \ldots, m_k)$, over a finite field F. The k secrets and the z random keys, $(u_1, \ldots, u_z)$, define a polynomial of degree $z + k - 1$. Evaluating $p(x)$ over n distinct non-zero points $(x_1, \ldots, x_n)$, yields n shares $c_i = p(x_i)$. Decoding the secret requires $z + k$ shares, with which the polynomial is reconstructed by interpolation.

Like in Shamir's original scheme, the polynomial is typically evaluated via multiplication by a $n \times (z + k)$ matrix, as depicted in Figure 2.2 (b). 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. This variation of Shamir's 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 and decode $n - r$ 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 $n - r - z$ [42]. 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 $\frac{k+r+z}{k}$.

2.2.3 AONT-RS

All-or-Nothing Transform with Reed-Solomon (AONT-RS) [78] 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 2.3, 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

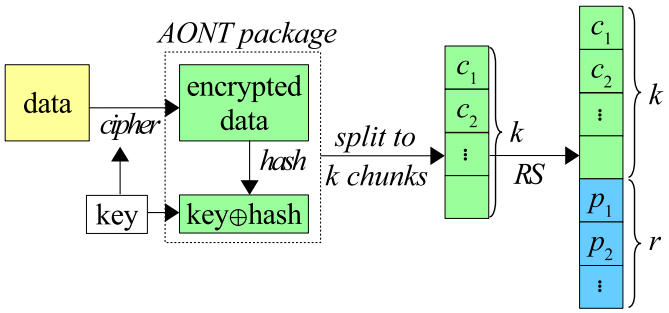

Figure 2.3: Encoding process of AONT-RS, $c_1, \ldots, c_k$ are the encrypted data chunks and $p_1, \ldots, p_r$ are the parity chunks.

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.

A known drawback of encryption is that it eliminates duplicates in the encrypted data. As a result, storage reduction techniques such as deduplication do not work on encrypted data. A variation of AONT-RS scheme, that allows deduplication, was presented in CDStore [57]. In this version, instead of using a random encryption key, the key is generated based on the object's content. Thus, identical data has an identical encryption key and the encrypted data is identical as well. This allows deduplication by both clients and servers.

The evaluation of AONT-RS in the original paper shows that this scheme is superior to secret sharing schemes. However AONT-RS was only compared to the basic Shamir's secret-sharing scheme, which is less efficient than the generalized version. Furthermore, the paper was published before hardware accelerated encryption was available, and before the introduction of efficient secret sharing schemes.

2.2.4 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

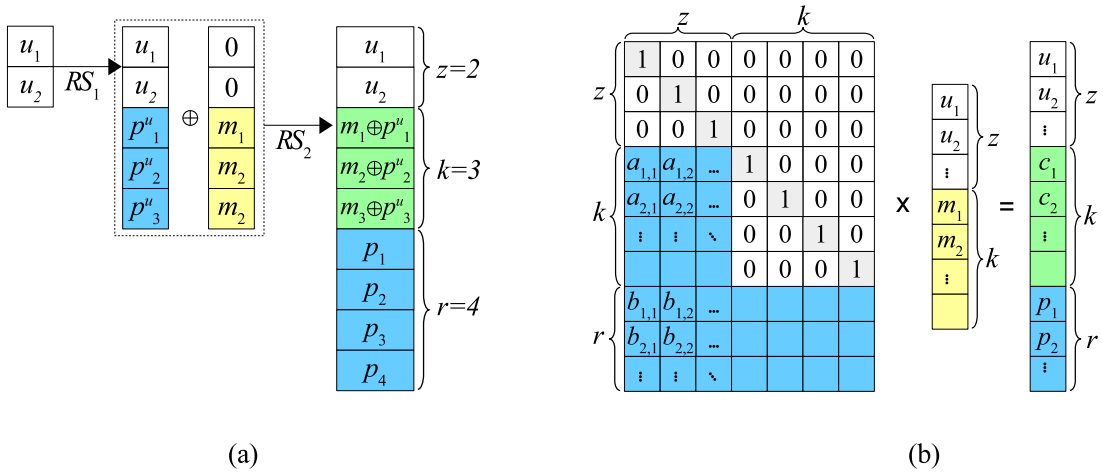

Figure 2.4: Encoding process of $(n = 9, k = 3, r = 4, z = 2)$ secure RAID (a) using two Reed-Solomon codes, and encoding process of (n, k, r, z) secure RAID (b) using generator $n \times (z + k)$ matrix.

when the erasure code used in the first step is a subcode of the erasure code from the second step. This ensures that the parity generated in the second step will not reveal more information on the secrets than any other share, proof in the paper [39].

Figure 2.4 (1) 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. 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. The encoding can also be done using multiplication of a *near-systematic* $n \times (z + k)$ generator matrix by a vector of keys and data elements, as depicted in Figure 2.4 (b).

Alternative constructions or secure RAID are based on array codes such as EVEN-ODD. Table 2.1 summarizes known constructions and their constraints on k,r, and z. We will use the construction based on Reed-Solomon code, which does not impose any constraint on k, r, and z, and is easy to build using existing implementations of Reed-Solomon code.

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

Erasure code	k	z	r	n	XOR only	Comment
Reed-Solomon	Any	Any	Any	$k+r+z$	No	Variety of implementations. When $r=z=1$ the scheme will work only for even k.
EVENODD [19]	$p-2$	2	2	$p+2$	Yes	p is prime, limitation of the code. k can be extended to other values by addition of virtual nodes, this complicates encoding and decoding.
STAR [38]	$p-3$	3	3	$p+3$	Yes	p is prime, limitation of the code.
RDP [23]	$p-3$	2	2	$p+1$	Yes	p is prime, limitation of the code.
B-codes [94]	2	2	2	6	Yes	Only one parameter set possible.

Table 2.1: Available constructions of secure RAID and their constraints on the scheme parameters k,r and z.

$n-r$ shares in existing secret-sharing schemes (note that typically, $n-r$ is considerably greater than z).

2.3 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 [20, 25, 32, 34, 35]. 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 [45].

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.

2.4 Challenges and Goals

The schemes described above have been designed with different objectives and trade-offs 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. 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 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.

Chapter 3

Computational overheads

3.1 Evaluation Goals

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.

3.2 Methodology

We implemented all the data protection schemes in C++ for scheme performance evaluation and in Java for the distributed objects store described in Chapter 4. Whenever possible, we based our implementation on existing verified and optimized implementations of standard procedures. For Reed-Solomon and matrix multiplications over finite fields, we used Jerasure library [74], which enhances finite field operations using vectorization, i.e SIMD instructions. We used only finite field operations over $GF(2^8)$, where each byte is an element in the field, this allows efficient implementation and convenience of working in byte granularity.

3.2.1 Cryptographic Functions

We used the OpenSSL cryptographic library [6], for all ciphers and cryptographic hash function implementations.

Component	Implementation	Provider	Comments
True RNG	/dev/random	Linux	Environmental noise as random source, including interrupts and RdRand
	RdRand	Intel	Thermal noise as random source
CSPRNG	/dev/urandom	Linux	Based on ChaCha, seeded periodically by the OS
	AES	OpenSSL (C++), SunJCE (Java)	AES 256 counter mode
PRNG	rand()	<cstdlib>	Not secure
	XOR	xoroshiro128+	
Hashing	MD5	OpenSSL (C++), Sun (Java)	128 bit hash
	SHA-1		160 bit hash
	SHA-256		256 bit hash
Symmetric key encryption	ChaCha	OpenSSL (C++), Bouncy Castle (Java)	Stream cipher, 128 bit keys, used in TLS [26, 53]
	AES	OpenSSL (C++), SunJCE (Java)	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	Our implementation (C++/Java)	Uses Jerasure for finite field operations in C++
	Secure RAID	Our implementation (C++/Java)	Based on Reed-Solomon

Table 3.1: Implementation details of the data protection primitives and schemes used in our evaluation.

Symmetric-key ciphers. We examined two different ciphers. ***ChaCha*** is a stream cipher [13] used in various secure communication protocols, such TLS [53]. ***AES*** is a popular symmetric-key block cipher [24], which is stronger than ChaCha, available in various modes of operation. We used AES in *counter (CTR)* mode, in which the data is XORed with a stream of values produced by encrypting successive values of a counter. The security level of the cipher is determined by the size of the key, where 256 bits is the strongest and 128 bits is minimal and thus, weaker. AES cipher also has performance advantage as it can be accelerated in hardware via instruction set AES-NI [31] in x86 architecture. AES processor instructions is also available in other architectures, such as ARM [3], Oracles SPARC [10] and IBM's POWER7+ [17].

Cryptographic hash functions. We considered three cryptographic hash functions. ***MD5*** generating a 128-bit hash value, ***SHA-1*** generating a 160-bit hash value and ***SHA-256*** generating a 256-bit hash value. These hash functions are widely used for message authentication codes [49].

3.2.2 Random Number Generators

We examined several pseudo-random number generators, both secure and non-secure. We seeded all generators with values with true random data from `/dev/random`, although the size of the seed differed for different generators. We used two CSPRNGs, based on different cryptographic functions for secure random generation. Our ***AES*** CSPRNG

implementation uses AES-256 in CTR mode, initialized with a 256-bit key and a random counter. `/dev/urandom` is native CSPRNG in Linux, it is based on ChaCha in Linux kernel 4.9. The generator is seeded periodically by the operating system. This is considered a vulnerability—users cannot verify that the output does not depend on previous output [32].

We also employed two true random number generators available on our evaluation setup. First is Linux's `/dev/random`, which is seeded constantly by the operating system. ***RdRand*** is Intel's digital random number generator, seeded using thermal noise in the chip.

For evaluation purposes, we used two non-cryptographic PRNGs. We denote ***rand()*** the basic PRNG supplied in <cstdlib> in C++. ***Xoroshiro128+ (XOR)*** is the fast PRNG [90].

We also use a "fake" PRNG, ***None***, which reads data from a predefined array in memory. This served as our baseline for evaluating the effect of random data generation on the throughput of the schemes that require it.

3.2.3 Implementation of Data Protection Schemes

For encryption, we used AES-256 and ChaCha. We generated keys from `/dev/random`, and stored them locally for decryption. Secure key management is outside the scope of this evaluation. After the data was encrypted parity chunks were constructed using RS erasure code. For AONT-RS, we used AES-128 (for encryption) and SHA-1 (for the cryptographic hash), as these were the fastest combination available. For the secret-sharing schemes, we used the PRNGs specified above. We implemented Shamir's scheme and its generalization using finite field matrix multiplication in Jerasure. Our secure RAID implementation is based on the Reed-Solomon implementation from Jerasure library as well.

Implementation details of the data protection primitives and schemes are summarized in Table 3.1.

3.2.4 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 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.

	Ciphers			Hash functions (digest)		
	ChaCha	AES-128	AES-256	MD5	SHA-1	SHA-256
Encrypt	841.54	4537.93	3379.98	606.65	853.73	377.9
Decrypt	846.62	4569.03	3425.19			

Table 3.2: Measured throughput (MB/s) of cryptographic functions. For ciphers, encryption and decryption throughput; for hash functions, digest throughput. AES has significantly higher throughput thanks to hardware acceleration.

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 3.3: Measured throughput (MB/s) of random data generation. True random data generation is too slow for anything but seeding. AES secure PRNG is fastest thanks to hardware acceleration.

• **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 each scheme according to its properties.

3.3 Results

3.3.1 Cryptographic Function Overhead

For the ciphers in our schemes, we measured the encryption and decryption throughput, and for the hash functions we measured the digest throughput. Our results, summarized in Table 3.2, show that AES achieves a speedup of up to 5x compared to Chacha, thanks to its hardware acceleration.

3.3.2 Random Number Generation

We measured the throughput of six RNGs detailed in Table 3.1. Our results, summarized in Table 3.3, 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, even more than the non-secure PRNGs, thanks to hardware 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 performance. Figure 3.1 shows the results for $k = 2, 8, 32$ and $r = z = 2$. Our results 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.

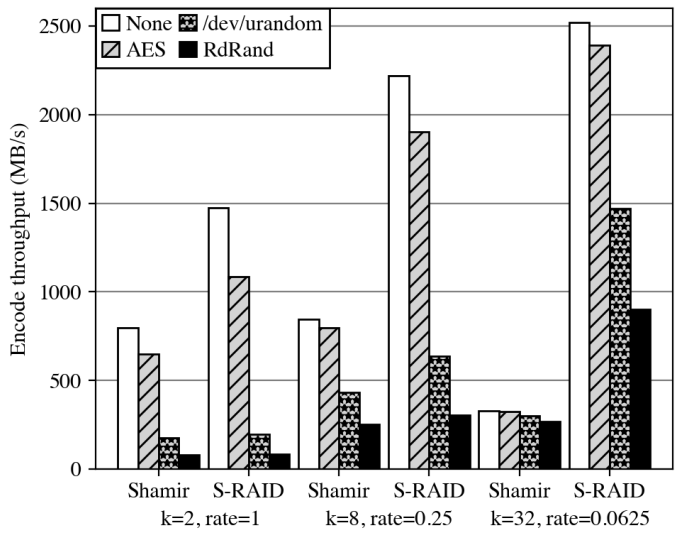

Figure 3.1: Effect of random data generation on secret-sharing schemes with different random rates and $r = z = 2$. Using hardware accelerated secure PRNG minimizes the overhead.

To reason about these results, we define the *random rate* as $\frac{z}{k}$, the ratio between the amount of random and data bytes in a stripe. Both schemes had the same random rate. Indeed, when $k = 2$ and the random rate was 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 encoding throughput by 3x. In the rest of our evaluation we used only AES CSPRNG. 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.

3.3.3 Encode/Decode Performance

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 computational overhead and outperforms the other schemes. (3) AONT-RS performance is limited by the cryptographic hash function.

Figure 3.2 shows encode (a) decode (b) throughput of all schemes with $r = z = 2$ and different k values. 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. AES scheme encoding throughput is lower (2160 MB/s), than AES cipher encryption throughput (3380 MB/s in Table 3.2), as the scheme includes Reed-Solomon encoding as well. AONT-RS had the lowest encoding and decoding throughput, about 650 MB/s. This is due to the overhead of hash calculation, which is

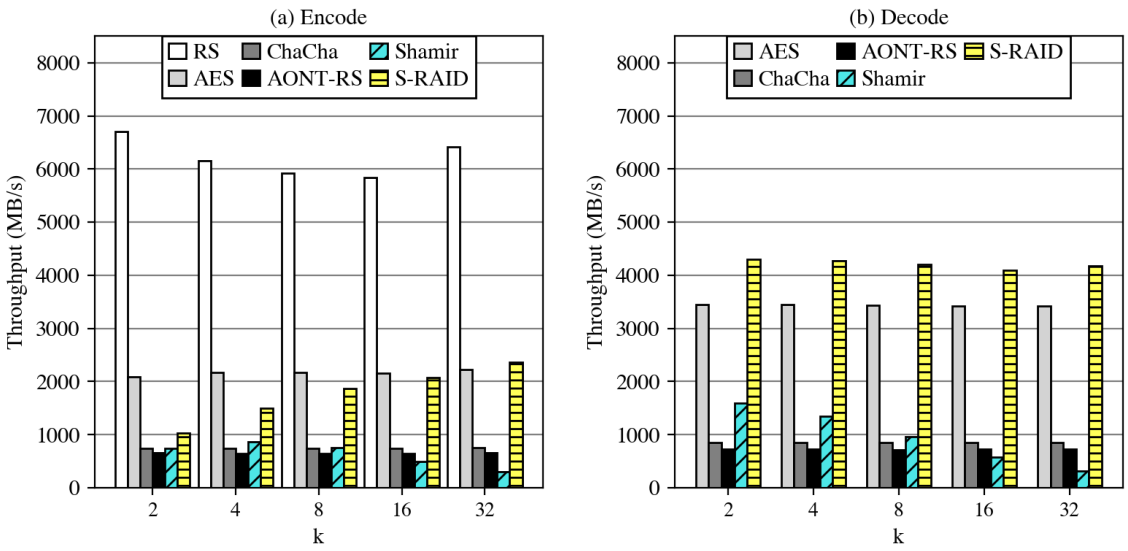

Figure 3.2: Encoding (a) and decoding (b) with $r = z = 2$. The high overhead of encryption is eliminated by hardware acceleration. AONT-RS suffers the overhead of non-accelerated cryptographic hash function. Shamir's high overhead prevents its throughput from increasing with k, despite the decrease in random rate. In decoding secure RAID outperforms all the schemes, thanks to its near-systematic encoding.

prohibitive because of its low throughput.

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 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 in which parity generation was responsible for more of the overall overhead.

Figure 3.3 (a) shows encode throughput of all schemes with with $r = 1$ and $z = 2$. Encoding throughput increased by over 100% for Reed-Solomon and about 30% for AES and secure RAID, and by 6% for ChaCha and for AONT-RS. In Shamir's scheme, the relative weight of one parity generation decreased with increase in k, due to its non-systematic encoding of the remaining $n - 1$ shares. Its encoding throughput increased

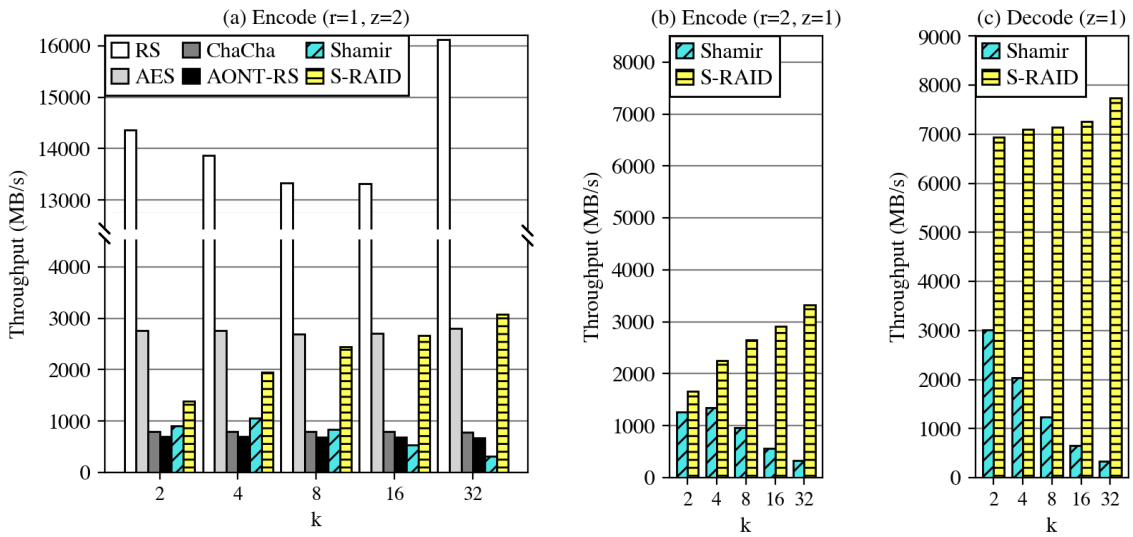

Figure 3.3: Encoding with $r = 1$, $z = 2$ (a) and with $r = 1$, $z = 2$ (b); and decoding with $z = 1$ (c). Changing r does not influence systematic decoding, and changing z is only applicable for secret-sharing schemes.

by to 22% with $k = 2$ and only by 4% with $k = 32$.

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.

Figure 3.3 shows encode (b) and decode (c) throughput of all schemes with $r = 2$ and $z = 1$. Secure RAID encoding and decoding throughput increased by about 45% and 70% respectively. In Shamir's scheme encoding and decoding complexity depends on k, thus the influence of reducing z decreased with increase in k. Its encoding throughput increased by 9% ($k = 32$) to 70% ($k = 2$), and decoding throughput increased by 6% ($k = 32$) to 85% ($k = 2$).

Degraded Decode Performance

Figure 3.2 (c) shows the decode throughput of each scheme when two systematic shares are unavailable, and $r = z = 2$. Reed-Solomon reconstruction stands as baseline to other schemes, as almost all of them include Reed-Solomon reconstruction as part of degraded decode process.

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 the affect decoding method. The throughput of degraded decode with secure RAID was roughly half that of regular decode. The throughput increased slightly

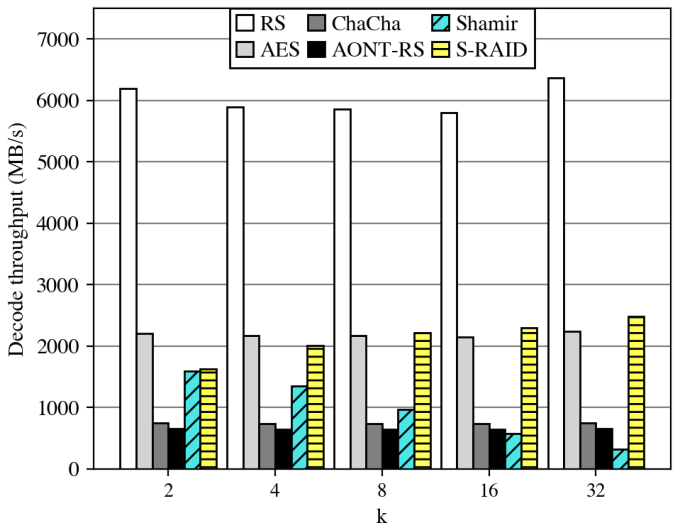

Figure 3.4: Degraded decode throughput with $r = z = 2$ and two unavailable systematic shares. Data recovery affects only schemes with efficient decoding.

with an increase in k, as the relative portion of reconstructed shares decreased.

Random Access Decode Performance

Figure 3.5 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, still as size of the share decreased the scheme had less data to decode, and thus the decode latency decreased as well. Secure RAID, on the other hand, required only $z+1$ shares to decode a single share, and it achieved fastest random access decoding, 16–30% faster than AES.

Conclusions

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.

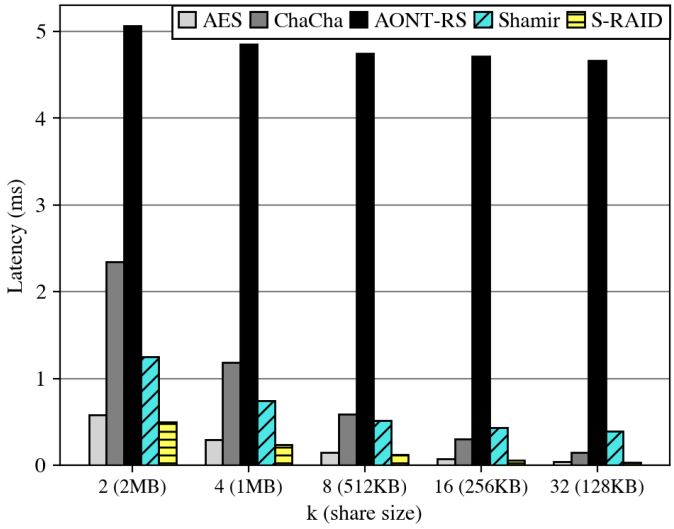

Figure 3.5: Random access average decode latency with $r = z = 2$. Random access performance is a major drawback of AONT-RS.

Chapter 4

End-to-End Evaluation

4.1 Evaluation Goals

In the previous chapter, 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 and different storage types.

4.2 Methodology

4.2.1 Object Store Implementation

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. As a result, we re-implemented all our data protection schemes in Java. (see details in Table 3.1).

For consistency, we compared the single threaded encoding and decoding throughput of the data schemes in Java and in C++. Table 4.1 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

	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 4.1: 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.

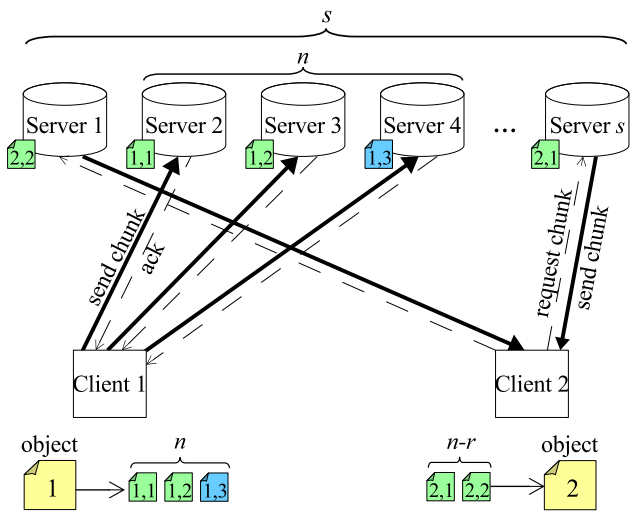

Figure 4.1: A high-level illustration of our object store, with write and read operations. In the write, object is encoded creating n shares, which are sent to n consecutive servers. In read, $n-r$ shares are requested from the servers on which they reside and the object is decoded from these shares.

modules. To ensure that encoding and decoding are not the bottleneck in our LAN and multi-cloud 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 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.

Figure 4.1 depicts the high-level representation of the object store and its main operations to write and read objects as described earlier. The client is connected to s servers, shares of each object are distributed to n different servers. For object read the client requests the systematic shares from $n-r$ servers, in case of encryption based scheme $n-r=k$ and in case of secret-sharing scheme $n-r=k+z$.

Algorithm 1 presents the pseudocode of the write operation executed by the client. The main thread only reads the objects and submits them to the thread pool for

Algorithm 1 Client operation on write object

```
function writeObject(object, servers, codec, n, k, z)
    obj_shares = codec.encodeObject(object)                ▷ executed via thread pool
    obj_servers =getObjectServers(object.ID, servers, n)   ▷ pseudocode below
    for all server, share ∈ obj_servers, obj_shares do
        server.pushShareToSendQ(share)                     ▷ shares are sent asynchronously
    end for
end function
```

encoding. After the shares are created by a thread in a thread pool, the shares are pushed to the send queue of the appropriate server socket.

Algorithm 2 Client operation on object read

```
function readObject(object_ID, servers, codec, n, k, z)
    obj_servers =getObjectServers(object_ID, servers, (n − r))
    for all server ∈ obj_servers do
        server.sendRequest(object_ID)   ▷ each server contains at most one share
    end for
end function
upon event share received from server do
    shares.append(share)
    if len(shares) ≥ (n − r) then
        object = codec.decodeObject(shares)                ▷ executed via thread pool
    end if
```

Algorithm 2 contains pseudocode of the basic read objects operation. First $n - r$ systematic shares of the objects are requested from the appropriate servers. Then in an asynchronous event handler after the $n - r$ shares received from servers, object is submitted for decoding via thread pool. For degraded decode we select up to r servers to be unavailable and request the $n - r$ shares only from available servers. Random share read is implemented slightly differently, as each scheme requires to read different number of shares per for single share decode. The only difference is in the selection of servers from which to read which is delegated to the *codec* object, and implemented for each scheme separately.

Algorithm 3 Getting servers that store objects shares, computes first server and the rest are subsequent servers, in an round-robin fashion.

```
function getObjectServers(object_ID, servers, shares_num)
    servers_num =len(servers)
    obj_servers = ∅
    for 0 ≤ i < shares_num do
        server = (i + object_ID · n) mod servers_num
        obj_servers.append(servers[server])
    end for
    return obj_servers
end function
```

Algorithm 3 contains our method of selecting servers for storage of object shares. We calculate the server that stores the first share and other shares are stored in a subsequent servers in a round-robin fashion.

4.2.2 Evaluation Setup

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. For optimized load balancing, we further ensured that $\gcd(s, n) = 1$. 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 client and server caches[1].

LAN Setup

Our local cluster used five machines identical to the one described in Section 3, connected by a 10Gb Ethernet network and equipped with four Dell 960GB SATA SSDs. 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 Setup

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 [7]:

- `c4.large` had two virtual CPUs, 3.75 GiB of RAM and "moderate network bandwidth".
- `c4.xlarge` had four virtual CPUs, 7.5 GiB of RAM and "high network bandwidth".

We configured our servers with three storage types:

- The *General Purpose SSD* is the default storage provided by Amazon Web Services (AWS), with baseline throughput of 100 IOPS.
- The *Provisioned IOPS SSD* provided 50 IOPS per 1 GB. We created volumes of 50

[1] We do this by applying two simple commands: `sudo echo 3 | sudo tee /proc/sys/vm/drop_caches && sudo sync`

GB, with 2500 IOPS per volume.

- The *Throughput Optimized HDD* supported 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).

We note that our our client machine was located in Israel, which is connected to Europe by optical fiber cables [8].

4.3 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 when it comes to small random accesses.

4.3.1 Write/Read Throughput

LAN Performance

Figure 4.2 shows 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, because less data is read than written per stripe and reading speed in SSD is generally faster, ChaCha and AONT-RS schemes reached their maximal read throughput with $k = 4$. 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 $k \leq 4$. With $k = 16$, the throughput of secure RAID was about 10% lower than that of AES. This was roughly the difference between the storage overhead of those schemes. Encryption

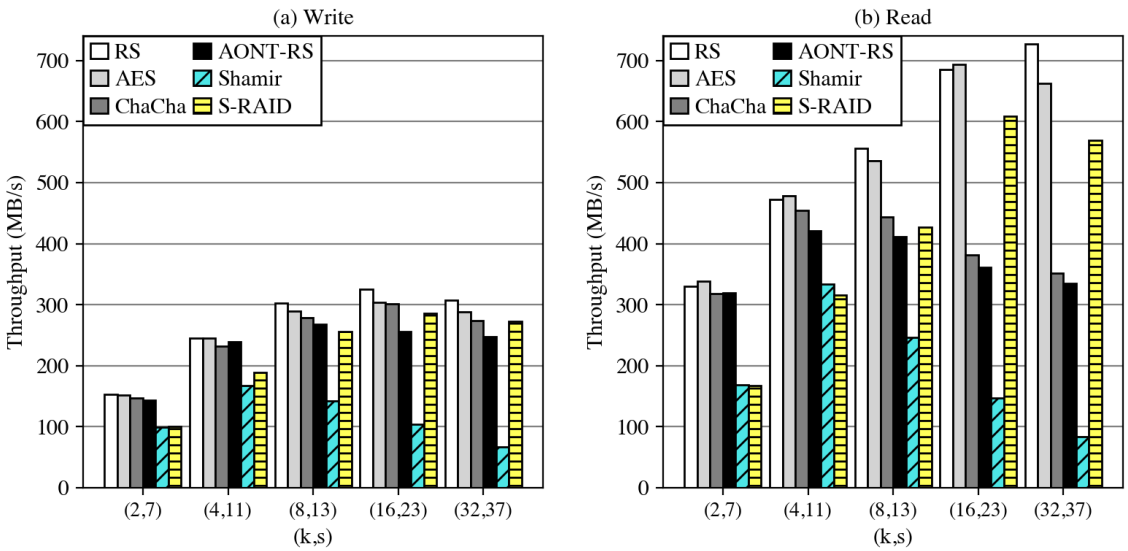

Figure 4.2: Write (a) and read (b) throughput in the LAN setup with $r = z = 2$. I/O throughput becomes the bottleneck of all schemes except Shamir's secret sharing. Computation remains the bottleneck for ChaCha, AONT-RS, and Shamir's scheme.

wrote $n = 18$ shares and read $k = 16$ per object, while secure RAID wrote $n = 20$ and read $k + z = 18$ shares.

We repeated the write measurements with different r and z combinations. The results showed similar trends to those in Figure 4.2. The effect of reducing both r and z was similar to the effect this had on encoding throughput, yet for different reason. Reducing r from 2 to 1 increased the throughput of all schemes due to the reduced storage overhead. The increase was lower in Shamir's scheme for $k \geq 4$, where encoding was the limiting factor. With $k = 2$, the reduction in r increased throughput of all of the schemes by 30-33%. Reducing z from 2 to 1 had a similar effect on both secret sharing schemes.

Multi-Cloud Performance

Figure 4.3 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 $k = 16$, 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

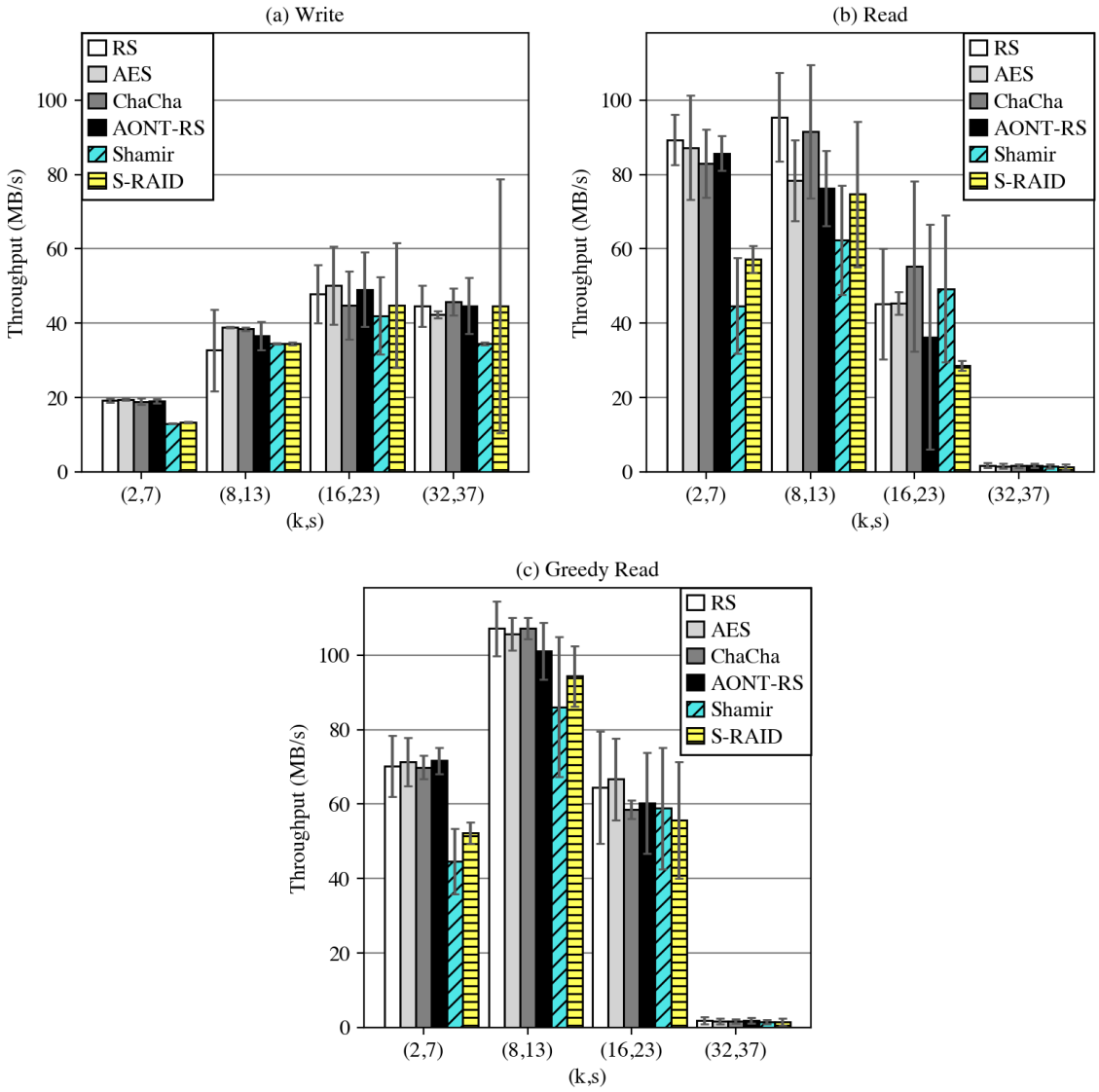

Figure 4.3: Write (a), read (b) and greedy read (c) throughput in multi-cloud setup, on `c4.xlarge` instances with general purpose SSD storage and $r = z = 2$. In multi-cloud environments, the network bandwidth dominates performance. The amount of redundancy (r) determines the number of high-latency servers the system can tolerate.

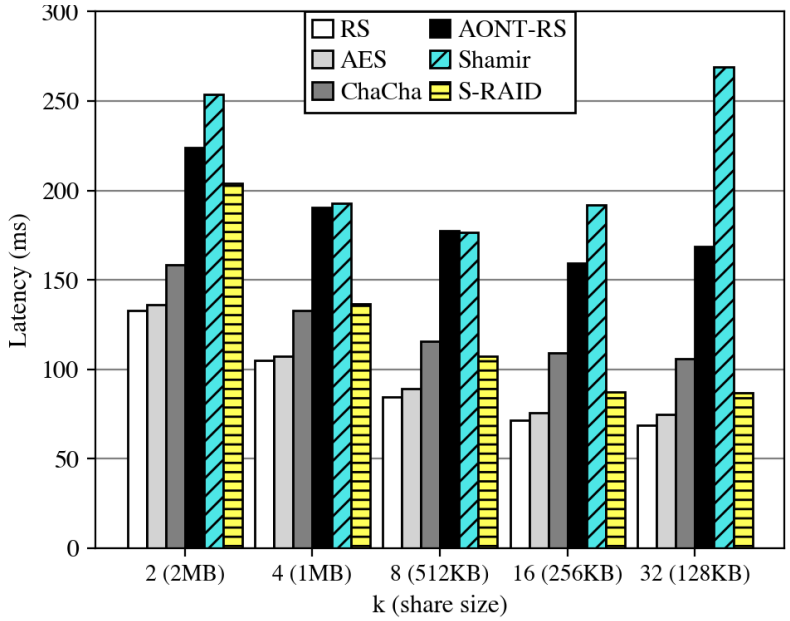

Figure 4.4: Random access latency in LAN setup with $r = z = 2$. AONT-RS and Shamir's scheme require k and $k + z$ shares respectively for decoding a single chunk.

($s = 37$) 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.

4.3.2 Random Access Latency

Figure 4.4 shows the average latency of all schemes when reading one share from a stripe, with $r = z = 2$ 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 key shares to decode the data share, while the other schemes read only one. 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.

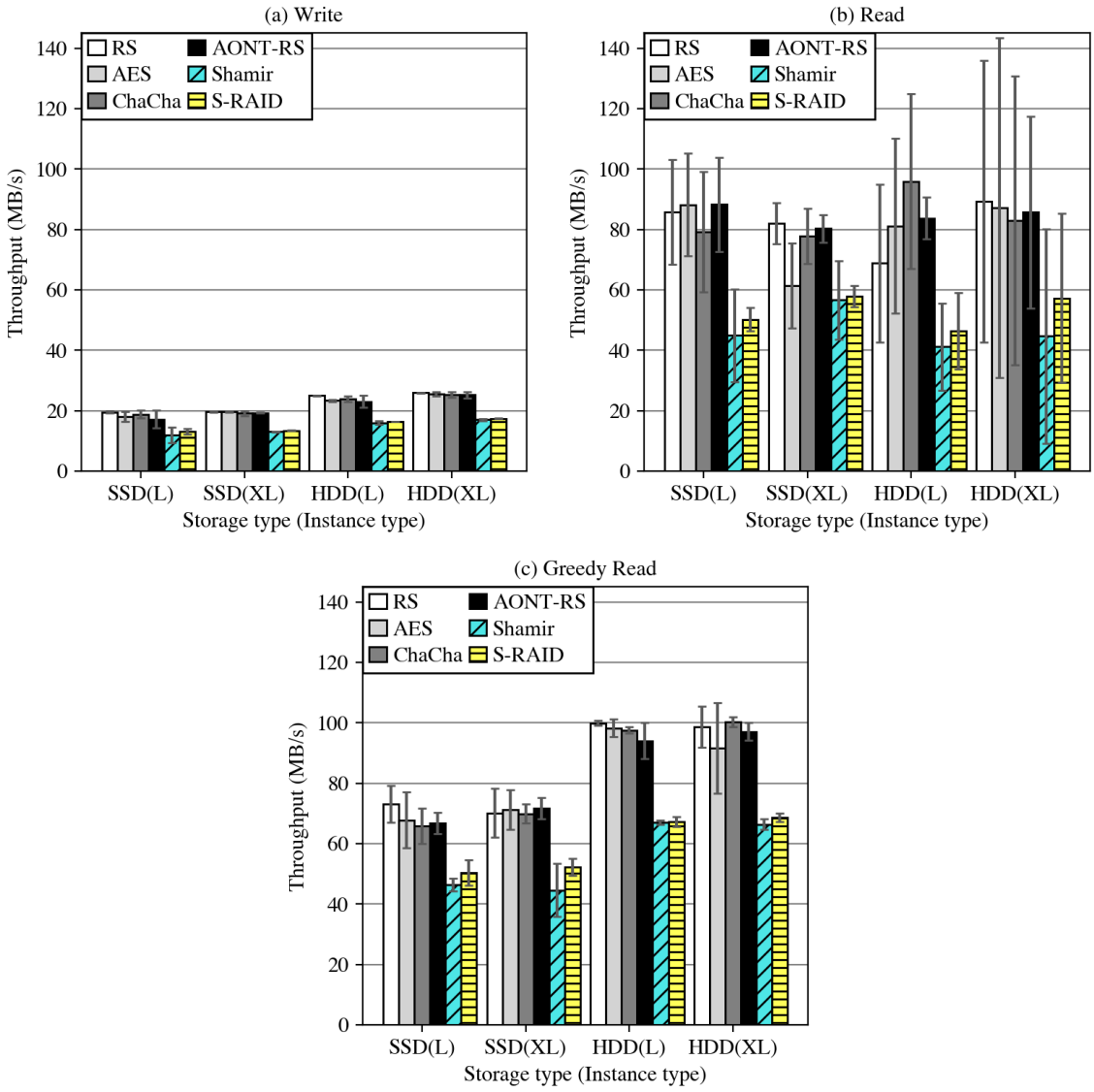

Figure 4.5: Write (a), read (b) and greedy read (c) throughput in multi-cloud setting with $k = r = z = 2$ and two storage and instance types. 'L ' is `c4.large` and 'XL' is `c4.xlarge`. Network is the bottleneck in regular reads, but HDDs improve the throughput of write and greedy read operations.

4.3.3 Storage and Server Type

We repeated the experiment in the small multi-cloud ($s = 7$) with all combinations of machine and storage types. The results for the Provisioned IOPS SSD were identical to those of the General Purpose SSD, and thus we omit them from our discussion.

Figure 4.5 shows the write, read, and greedy read performance with instances deployed on machines with moderate (`c4.large`) and fast (`c4.xlarge`) network connection, with SSD and HDD storage.

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.

4.3.4 Conclusions

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.

Chapter 5

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.

5.1 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. Several solutions to this problem entail increased storage overhead [9, 70, 80, 82, 89] which, as our results indicate, will likely reduce read and write throughput. In POTSHARDS [87], a protocol for secure reconstruction is proposed, protecting the data in case $z = 1$, i.e. at most one eavesdropper. The proposed protocol offers reconstruction using simple parity, but can be modified to other types of parity. As part of the protocol, an additional random mask is transferred with every share, doubling the repair network cost. This protocol will not suffice if two or more nodes are compromised. Methods for minimizing this cost and general reconstruction protocol for any z are studied in [40, 47, 75, 76]. Reconstruction does not compromise the security of encryption based schemes in which the keys are managed in separate secure stores. This is done in Hybris [27], where the keys are stored together with the meta-data in a private cloud, which provides added security.

5.2 Deduplication

Storage service providers eliminate duplicate data from their systems in order to reduce storage and network costs [28, 29, 85, 95]. 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 [57, 86]. A similar solution can be applied to secret-sharing schemes [55].

However, our results indicate that this will significantly reduce encoding throughput, unless both encryption and hashing are hardware accelerated.

5.3 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 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.

5.4 Storage Types

Our evaluation provides some insight into the effect of several technological trends. As storage-class memory and RAM-based storage [68] gain popularity, the bottlenecks in the data path shift from storage to computation. In such architectures, the bottlenecks we identified in Section 3 may no longer be masked by high storage and network 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 3.

5.5 Device Types and Network Overhead

In our evaluation, we measured the computational overhead of the schemes and their overall throughput on an enterprise-class server machine with high-end CPU and a large memory. However, in the general case, data protection is performed on all types of devices, from hand-held devices and small single-board computers running in home appliance devices to special high-performance computing (HPC) machines processing data at petabyte-per-second rates.

When data protection is running on mobile device, which have limited RAM and processing power, the computational overhead may become critical. Furthermore, the computational overhead will directly influence power consumption (or battery lifetime), which is also a limited resource in such devices. Nevertheless, lately computational power of mobile devices increased and new hardware acceleration techniques for cryptographic functions [3] were introduced. Thus, the bottleneck is now the network bandwidth. Currently the fastest available cellular network throughput is up to 2.6 Gbit/s [2] per

cell (meaning it will be divided between all of the users in that cell's area), while the multi threaded encoding and decoding throughput of AES on new Qualcomm chipset is approximately 5 GB/s [81].

On a laptop device the computational overhead is less critical—large amounts of RAM and powerful CPUs with special instructions sets for hardware accelerated cryptographic functions are already a standard. The fastest available wireless network achieves a throughput of up to 1.3 Gbit/s per work station [4]. However, when the data is uploaded and downloaded in a realistic WAN configuration, the throughput will likely be much lower. For example, in a 100 Mbit/s Internet connection in Israel, we measured a download speed of 60-35 Mbit/s and an upload speed of 5-2 Mbit/s. Thus, for a client running on laptop in standard home environment the network will become the bottlenecks.

Chapter 6

Related work

To protect data in a distributed storage 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 [27, 33, 50]. A *consensus* mechanism ensures that file writes, updates, and deletes are only performed by authorized users on a threshold number of nodes. Authorized users are *authenticated* by a separate interface, which also verifies user permissions using tokens, access control lists, or other schemes [54, 62, 64]. 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 [26, 93]. 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 [50], Pond [79], and Glacier [33], addressed access control, serialized updates, load balancing, routing, and fault tolerance. They all assume the data has been encrypted prior to being distributed, while maintenance of encryption keys remains the responsibility of the clients. The encrypted data is encoded with Reed-Solomon erasure codes in order to ensure its durability in the face of large scale node failures.

Most multi-cloud architectures follow a similar approach. MetaStorage [12] addresses the durability of the data by replication, and relies on Byzantine agreement protocol for object updates. A slightly different approach is taken in Hybris [27]: metadata containing signatures of the data is replicated in a private and secure cloud, while the data is dispersed between multiple public clouds. This ensures strong consistency by leveraging strong consistency of metadata stored off-clouds to mask the weak consistency of data stored in clouds. DepSky [14] and SCFS [15] incorporate encryption into their client, along with a secret-sharing scheme for securely storing the encryption keys. In all these systems, erasure coding is performed on the encrypted data, as in our evaluation.

Several studies proposed that the storage overhead of secret-sharing schemes be reduced by reducing the capacity of individual shares. One approach is to store only the

seed of the randomly generated data, which requires regeneration of this data during the decoding process [88]. 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 [39, 41, 61]. In [52, 60] new secret-sharing schemes are proposed improving the computational complexity of basic Shamir's scheme, by requiring only binary (XOR) operations for encoding and decoding. BP-XOR [91] is another secret-sharing scheme constructed based on popular LDPC codes, with decoding executed using belief propagation technique, achieving only linear number of XOR operations for both encoding and decoding. Another approach is taken in SSMS (Secret-Sharing Made Short) [48] the information-theoretical security is sacrificed, achieving computational security instead. However, our results show that the cost of true random data generation is too high, due to the limited rate of measuring external noise, which also may require special hardware. Further, when encoding is performed on virtual machine that shares hardware, true random generation cannot be used safely [45]. Therefore, 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.

Chapter 7

Conclusions

We performed the first comprehensive comparison of encryption-based and secret-sharing schemes. Our evaluation shows that information-theoretical security is infeasible in real system implementations, due to the high cost of true random data generation. Thus, both approaches provide computational security. 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.

Bibliography

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

[2] NSN and Sprint achieves huge leap in TD-LTE network speeds. http://www.telecomtiger.com/Technology_fullstory.aspx?storyid=19712&section=S210, Feb. 2014.

[3] ARM® Cortex®-A57 MPCore Processor Cryptography Extension Technical Reference Manual. http://infocenter.arm.com/help/topic/com.arm.doc.ddi0514g/DDI0514G_cortex_a57_mpcore_cryptography_trm.pdf, 2015.

[4] Motorola Modular Access Points Performance Review. http://broadbandlanding.com/research/mmap/, 2017.

[5] 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/, Sept. 2017.

[6] OpenSSL. http://www.openssl.org, 2017.

[7] Amazon EC2 Instance Types. https://aws.amazon.com/ec2/instance-types/, 2018.

[8] Submarine Cable Map. https://www.submarinecablemap.com/, Feb. 2018.

[9] A. Agarwal and A. Mazumdar. Security in locally repairable storage. *Manuscript, arXiv:1503.04244*, 2015.

[10] D. Anderson. SPARC T4 OpenSSL Engine. https://blogs.oracle.com/solaris/sparc-t4-openssl-engine-v2, 2015.

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

[12] D. Bermbach, M. Klems, S. Tai, and M. Menzel. MetaStorage: A Federated Cloud Storage System to Manage Consistency-Latency Tradeoffs. In *IEEE International Conference on Cloud Computing (CLOUD '11)*, July 2011.

[13] D. J. Bernstein. Chacha, a variant of salsa20. In *Workshop Record of SASC*, 2008.

[14] A. Bessani, M. Correia, B. Quaresma, F. André, and P. Sousa. Depsky: Dependable and secure storage in a cloud-of-clouds. *Trans. Storage*, 9(4):12:1–12:33, Nov. 2013.

[15] A. Bessani, R. Mendes, T. Oliveira, N. Neves, M. Correia, M. Pasin, and P. Verissimo. Scfs: A shared cloud-backed file system. In *USENIX Annual Technical Conference (ATC '14)*, June 2014.

[16] G. R. Blakley and C. Meadows. Security of ramp schemes. In *Workshop on the Theory and Application of Cryptographic Techniques*, 1984.

[17] B. Blaner, B. Abali, B. M. Bass, S. Chari, R. Kalla, S. Kunkel, K. Lauricella, R. Leavens, J. J. Reilly, and P. A. Sandon. Ibm power7+ processor on-chip accelerators for cryptography and active memory expansion. *IBM Journal of Research and Development*, 57(6):3:1–3:16, Nov 2013.

[18] M. Blaum, J. Brady, J. Bruck, and J. Menon. EVENODD: an efficient scheme for tolerating double disk failures in RAID architectures. *IEEE Transactions on Computers*, 44(2):192–202, 1995.

[19] M. Blaum and R. M. Roth. On lowest density MDS codes. *IEEE Transactions on Information Theory*, 45(1):46–59, 1999.

[20] M. Bucci, L. Germani, R. Luzzi, A. Trifiletti, and M. Varanonuovo. A high-speed oscillator-based truly random number source for cryptographic applications on a smart card IC. *IEEE Transactions on Computers*, 52(4):403–409, 2003.

[21] D. Burihabwa, P. Felber, H. Mercier, and V. Schiavoni. 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)*, 2016.

[22] C. Cachin, R. Haas, and M. Vukolic. Dependable storage in the intercloud. Technical Report RZ 3783, IBM, May 2010.

[23] P. Corbett, B. English, A. Goel, T. Grcanac, S. Kleiman, J. Leong, and S. Sankar. Row-diagonal parity for double disk failure correction. In *USENIX Symposium on File and Storage Technologies (FAST '04)*, 2004.

[24] J. Daemen and V. Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer Science & Business Media, 2013.

[25] D. Davis, R. Ihaka, and P. Fenstermacher. Cryptographic randomness from air turbulence in disk drives. In *Annual International Cryptology Conference on Advances in Cryptology (CRYPTO '94)*, 1994.

[26] T. Dierks. The transport layer security (tls) protocol version 1.2. 2008.

[27] D. Dobre, P. Viotti, and M. Vukolic. Hybris: Robust hybrid cloud storage. In *Annual ACM Symposium on Cloud Computing (SOCC '14)*, November 2014.

[28] F. Douglis, A. Duggal, P. Shilane, T. Wong, S. Yan, and F. Botelho. The logic of physical garbage collection in deduplicating storage. In *USENIX Conference on File and Storage Technologies (FAST '17)*, 2017.

[29] M. Fu, D. Feng, Y. Hua, X. He, Z. Chen, W. Xia, F. Huang, and Q. Liu. Accelerating restore and garbage collection in deduplication-based backup systems via exploiting historical information. In *USENIX Annual Technical Conference (ATC '14)*, 2014.

[30] V. Goel and N. Perlroth. Yahoo says 1 billion user accounts were hacked. `https://www.nytimes.com/2016/12/14/technology/yahoo-hack.html`, Dec. 2016.

[31] S. Gueron. Intel® advanced encryption standard (aes) new instructions set. *Intel Corporation*, 2010.

[32] Z. Gutterman, B. Pinkas, and T. Reinman. Analysis of the linux random number generator. In *IEEE Symposium on Security and Privacy (S&P '06)*, May 2006.

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

[34] M. Hamburg, P. Kocher, and M. E. Marson. Analysis of intel's ivy bridge digital random number generator. Technical Report, Cryptography Research Inc, Mar. 2012.

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

[36] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin. Erasure coding in Windows Azure storage. In *USENIX Annual Technical Conference (ATC '12)*, 2012.

[37] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, S. Yekhanin, et al. Erasure coding in windows azure storage. In *USENIX Annual Technical Conference (ATC '12)*, 2012.

[38] C. Huang and L. Xu. STAR : an efficient coding scheme for correcting triple storage node failures. In *USENIX Conference on File and Storage Technologies (FAST '05)*, 2005.

[39] W. Huang and J. Bruck. Secure raid schemes for distributed storage. In *IEEE International Symposium on Information Theory (ISIT '16)*, July 2016.

[40] W. Huang and J. Bruck. Generic secure repair for distributed storage. *CoRR*, abs/1706.00500, 2017.

[41] W. Huang and J. Bruck. Secure raid schemes from evenodd and star codes. In *IEEE International Symposium on Information Theory (ISIT '17)*, June 2017.

[42] W. Huang, M. Langberg, J. Kliewer, and J. Bruck. Communication efficient secret sharing. *CoRR*, abs/1505.07515, 2015.

[43] W. Huang, M. Langberg, J. Kliewer, and J. Bruck. Communication efficient secret sharing. *IEEE Transactions on Information Theory*, 62(12):7195–7206, 2016.

[44] J. Kastrenakes. 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`, Mar. 2017.

[45] B. Kerrigan and Y. Chen. A study of entropy sources in cloud computers: random number generation on cloud hosts. *Computer Network Security*, pages 286–298, 2012.

[46] O. Khan, R. C. Burns, J. S. Plank, W. Pierce, and C. Huang. 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)*, 2012.

[47] A. S. R. Koyluoglu, Onur Ozan and S. Vishwanath. Secure cooperative regenerating codes for distributed storage systems. *IEEE Transactions on Information Theory*, 60(9):5228–5244, Sept 2014.

[48] H. Krawczyk. Secret sharing made short. In *Annual International Cryptology Conference on Advances in Cryptology*, 1994.

[49] H. Krawczyk, R. Canetti, and M. Bellare. Hmac: Keyed-hashing for message authentication. 1997.

[50] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: An architecture for global-scale persistent storage. *SIGPLAN Not.*, 35(11):190–201, Nov. 2000.

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

[52] J. Kurihara, S. Kiyomoto, K. Fukushima, and T. Tanaka. A new (k, n)-threshold secret sharing scheme and its extension. In *International Conference on Information Security (ISC '08)*, 2008.

[53] A. Langley, W.-T. Chang, N. Mavrogiannopoulos, J. Strombergson, and S. Josefsson. ChaCha20-Poly1305 cipher suites for transport layer security (TLS). RFC 7905, June 2016.

[54] A. W. Leung and E. L. Miller. Scalable security for large, high performance storage systems. In *ACM Workshop on Storage Security and Survivability (StorageSS '06)*, 2006.

[55] J. Li, X. Chen, M. Li, J. Li, P. P. Lee, and W. Lou. Secure deduplication with efficient and reliable convergent key management. *IEEE Transactions on Parallel and Distributed Systems*, 25(6):1615–1625, June 2014.

[56] M. Li and P. P. Lee. 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)*, 2014.

[57] M. Li, C. Qin, and P. P. C. Lee. Cdstore: Toward reliable, secure, and cost-efficient cloud storage via convergent dispersal. In *USENIX Annual Technical Conference (ATC '15)*, 2015.

[58] M. Li, C. Qin, P. P. C. Lee, and J. Li. Convergent dispersal: Toward storage-efficient security in a cloud-of-clouds. In *USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage '14)*, Jun. 2014.

[59] Y. Li, N. S. Dhotre, Y. Ohara, T. M. Kroeger, E. Miller, and D. D. E. Long. Horus: Fine-grained encryption-based security for large-scale storage. In *USENIX Conference on File and Storage Technologies (FAST '13)*, 2013.

[60] C. Lv, X. Jia, L. Tian, J. Jing, and M. Sun. Efficient ideal threshold secret sharing schemes based on exclusive-or operations. In *International Conference on Network and System Security (NSS '10)*, Sept 2010.

[61] R. J. McEliece and D. V. Sarwate. On sharing secrets and reed-solomon codes. *Commun. ACM*, 24(9):583–584, Sept. 1981.

[62] E. L. Miller, D. D. E. Long, W. E. Freeman, and B. C. Reed. Strong security for network-attached storage. In *USENIX Conference on File and Storage Technologies (FAST '02)*, 2002.

[63] M. Nanavati, P. Colp, B. Aiello, and A. Warfield. Cloud security: A gathering storm. *Commun. ACM*, 57(5):70–79, May 2014.

[64] Z. Niu, K. Zhou, D. Feng, H. Jiang, F. Wang, H. Chai, W. Xiao, and C. Li. Implementing and evaluating security controls for an object-based storage system. In *IEEE Conference on Mass Storage Systems and Technologies (MSST '07)*, Sept 2007.

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

[66] D. O'Sullivan. Cloud Leak: How A Verizon Partner Exposed Millions of Customer Accounts. `https://www.upguard.com/breaches/verizon-cloud-leak`, 2017.

[67] D. O'Sullivan. The RNC Files: Inside the Largest US Voter Data Leak. `https://www.upguard.com/breaches/the-rnc-files`, 2017.

[68] J. Ousterhout, A. Gopalan, A. Gupta, A. Kejriwal, C. Lee, B. Montazeri, D. Ongaro, S. J. Park, H. Qin, M. Rosenblum, S. Rumble, R. Stutsman, and S. Yang. The RAMCloud storage system. *ACM Trans. Comput. Syst.*, 33(3):7:1–7:55, Aug. 2015.

[69] D. Pal, P. Khethavath, J. P. Thomas, and T. Chen. Multilevel threshold secret sharing in distributed cloud. In *International Symposium on Security in Computing and Communications (SSCC)*. 2015.

[70] S. Pawar, S. E. Rouayheb, and K. Ramchandran. Securing dynamic distributed storage systems against eavesdropping and adversarial attacks. *IEEE Transactions on Information Theory*, 57(10):6734–6753, Oct 2011.

[71] N. Perlroth. Yahoo says hackers stole data on 500 million users in 2014. `https://www.nytimes.com/2016/09/23/technology/yahoo-hackers.html`, Sept. 2016.

[72] J. S. Plank and M. Blaum. Sector-disk (SD) erasure codes for mixed failure modes in RAID systems. *Trans. Storage*, 10(1):4:1–4:17, Jan. 2014.

[73] J. S. Plank, K. M. Greenan, and E. L. Miller. Screaming fast Galois field arithmetic using Intel SIMD instructions. In *USENIX Conference on File and Storage Technologies (FAST '13)*, 2013.

[74] J. S. Plank, S. Simmerman, and C. D. Schuman. Jerasure: A library in c/c++ facilitating erasure coding for storage applications-version 1.2. 2008.

[75] K. Rashmi, N. B. Shah, K. Ramchandran, and P. V. Kumar. Regenerating codes for errors and erasures in distributed storage. In *IEEE International Symposium on Information Theory (ISIT '12)*, July 2012.

[76] A. S. Rawat, O. O. Koyluoglu, N. Silberstein, and S. Vishwanath. Optimal locally repairable and secure codes for distributed storage systems. *IEEE Transactions on Information Theory*, 60(1):212–236, Jan 2014.

[77] I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, 1960.

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

[79] S. C. Rhea, P. R. Eaton, D. Geels, H. Weatherspoon, B. Y. Zhao, and J. Kubiatowicz. Pond: the OceanStore prototype. In *USENIX Conference on File and Storage Technologies (FAST '03)*, 2003.

[80] B. Sasidharan, P. V. Kumar, N. B. Shah, K. Rashmi, and K. Ramachandran. Optimality of the product-matrix construction for secure MSR regenerating codes. In *International Symposium on Communications, Control and Signal Processing (ISCCSP '14)*, 2014.

[81] M. T. Serrafero. Qualcomm snapdragon 845 benchmarks. `https://www.xda-developers.com/qualcomm-snapdragon-845-hands-on-benchmarks-first-impressions/`, Feb. 2018.

[82] N. B. Shah, K. Rashmi, and P. V. Kumar. Information-theoretically secure regenerating codes for distributed storage. In *IEEE Global Telecommunications Conference (GLOBECOM '11)*, Dec 2011.

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

[84] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The Hadoop Distributed File System. In *IEEE Symposium on Mass Storage Systems and Technologies (MSST '10)*, May 2010.

[85] K. Srinivasan, T. Bisson, G. Goodson, and K. Voruganti. iDedup: Latency-aware, inline data deduplication for primary storage. In *USENIX Conference on File and Storage Technologies (FAST '12)*, 2012.

[86] M. W. Storer, K. Greenan, D. D. Long, and E. L. Miller. Secure data deduplication. In *ACM International Workshop on Storage Security and Survivability (StorageSS '08)*, 2008.

[87] M. W. Storer, K. M. Greenan, E. L. Miller, and K. Voruganti. POTSHARDS - a secure, recoverable, long-term archival storage system. *ACM Transactions on Storage*, 5(2):1–35, 2009.

[88] S. Takahashi and K. Iwamura. Secret sharing scheme suitable for cloud computing. In *IEEE International Conference on Advanced Information Networking and Applications (AINA '13)*, March 2013.

[89] R. Tandon, S. Amuru, T. C. Clancy, and R. M. Buehrer. Toward optimal secure distributed storage systems with exact repair. *IEEE Transactions on Information Theory*, 62(6):3477–3492, 2016.

[90] S. Vigna. Further scramblings of marsaglia's xorshift generators. *Journal of Computational and Applied Mathematics*, 315:175 – 181, 2017.

[91] Y. Wang. Privacy-preserving data storage in cloud using array bp-xor codes. *IEEE Transactions on Cloud Computing*, 3(4):425–435, Oct 2015.

[92] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn. Ceph: A scalable, high-performance distributed file system. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI '06)*, 2006.

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

[94] L. Xu, V. Bohossian, J. Bruck, and D. G. Wagner. Low-density MDS codes and factors of complete graphs. *IEEE Transactions on Information Theory*, 45(6):1817–1826, 1999.

[95] B. Zhu, K. Li, and H. Patterson. Avoiding the disk bottleneck in the Data Domain deduplication file system. In *USENIX Conference on File and Storage Technologies (FAST '08)*, 2008.

היתרון של סכימות חלוקת סוד הוא שהן לא משתמשות במפתחות הצפנה, אך הן סובלות מבעיות אחרות: תקורה באחסון הנתונים, קידוד ופענוח כבדים חישובית, וייצור כמויות גדולות של נתונים אקראיים לצורך הקידוד. לכן כיום סכימות אלו משמשות להגנה על ארכיונים ארוכי טווח של נתונים קרים או לקידוד של כמויות קטנות של נתונים, כמו מפתחות הצפנה.

AONT-RS הוצעה כחלופה לחלוקת סוד. שיטה זאת מבוססת הצפנה, אך במקום אחסון מפורש של מפתחות ההצפנה, המפתחות מעורבלים יחד עם הנתונים המוצפנים ומפוזרים בין שרתי אחסון עצמאיים. שיטה זו מאפשרת ל־AONT-RS להשיג ביצועים טובים יותר ותקורת אחסון נמוכה יותר מסכימות חלוקת סוד. גם בשיטה זאת רק מי שמשיג את כל החלקים יכול לפענח את הנתונים.

לאחרונה שני פיתוחים חדשים מבטיחים לחסל צוואר בקבוק עיקריים של הגנת על נתונים. הפיתוח הראשון הוא סכימת חלוקת סוד חדשה *secure RAID*, אשר ממזערת את התקורה החישובית של סכימות חלוקת סוד, ומאפשרת פענוח של מידע חלקי עבור קריאות אקראיות. פיתוח נוסף הוא האצת הצפנה בחומרה על ידי פקודות מעבד ייעודיות, ואימוץ של שיטה זאת בספריות קריפטוגרפיות פופולריות.

התרומה שלנו

ההתקדמות הטכנולוגית מציבה בפני מהנדסי מערכת שקלול תמורות חדש. הצפנה מספקת סודיות חישובית, אך דורשת יצירה וניהול של מפתחות הצפנה, ומסתמכת על מאיצים בחומרה לצורכי מימוש יעיל. סכימות חלוקת סוד מבטיחות סודיות מושלמת עם קידוד ופענוח יעיל, אך גוררות תקורה משמעותית באחסון נתונים. לרוע המזל, תוצאות המחקרים הקודמים לא נותנים תשובה איזו גישה עדיפה מבחינת ביצועי מערכת, מאחר והם נעשו לפני הפיתוחים האחרונים בשיטות אלו.

מטרת המחקר שלנו היא לגשר על הפער הזה על ידי השוואה של השיטות המתקדמות ביותר עבור שתי הגישות. אנו מציגים את הניתוח המקיף הראשון של סכימות חלוקת סוד והצפנה ומעריכים מחדש את שקלול התמורות במערכות אחסון מרוחק ומאובטח. ההערכה שלנו מתייחסת לכל השלבים בנתיב אחסון הנתונים, כולל יצירת נתונים אקראיים, יעילות חישובית של הקידוד וההצפנה, והתפוקה הכוללת של מערכת האחסון המבוזרת בסביבה מקומית ובסביבה מרוחקת.

בעבודה זו מימשנו שתי סכימות חלוקת סוד ושתי סכימות מבוססות הצפנה, ומדדנו את הביצועים שלהן במגוון רחב של פרמטרים במערכת, כולל השפעה של רמות זמינות ואבטחה והשפעה של התקני אחסון וארכיטקטורות רשת.

המסקנות העיקריות שלנו מהמחקר הן כדלקמן: (1) יצירת נתונים אקראיים איטית מדי לשילוב, לכן סודיות מושלמת לא ניתנת ליישום במערכות אחסון אמתיות. (2) סכימת Secure RAID פותרת לחלוטין את בעיית החישוב הכבדה של חלוקת סוד: רק הצפנה מואצת בחומרה מראה ביצועים טובים יותר. (3) כאשר לוקחים בחשבון את השפעות התקני האחסון והרשת, סכימות מבוססות הצפנה עדיפות על סכימות חלוקת סוד, בגלל תקורת אחסון נמוכה יותר. (4) רק הצפנה וסכימת Secure RAID מאפשרות קריאות אקראיות קטנות ביעילות.

תקציר

שירותי אחסון בענן מספקים ביצועים גבוהים, זמינות ועמידות לנפילות. הם גם מאפשרים שיתוף קבצים וגמישות, וכל זאת במחירים תחרותיים. העברת אחסון וניהול נתונים לענן יכולה להוזיל בצורה משמעותית את ההוצאות של מרכז אחסון נתונים פנים ארגוני, בלי לפגוע בביצועים וזמינות של מערכות האחסון.

עם זאת, עסקים ואנשים פרטיים רבים לא בוטחים בשירותים חיצוניים בשמירה על מידע רגיש. בעוד ששירותי אחסון באינטרנט יכולים להבטיח את עמידות הנתונים, הם לא יכולים להבטיח באופן מלא שמירה על סודיותו מפני עובד חברה זדוני או מושחת. דיווחים שפורסמו לאחרונה מראים כי מירב הספקים של שירותי ענן לא מציינים בתנאי השימוש שהמידע שייך למשתמש, ולא מספקים מנגנוני אבטחה כדי להגן על המידע. יתר על כן, מספר מקרים של "דליפת מידע" מהענן תועדו לאחרונה.

מגבלות נוספות של אחסון נתונים בענן מונעות אימוץ רחב יותר של שירותים אלו. אחת מהן היא נעילת ספק (vendor lock-in), בה המעבר בין ספק ענן אחד למשנהו (מסיבות עסקיות שונות) הופך להיות יקר מדי בשל עלות אחזור כמויות גדולות של נתונים או פיתוח ממשקים חדשים ליישומים. מגבלה נוספת מהווים נפילות ושיבושים אליהם חשופים כל ספקי אחסון בענן.

מודל אחסון חדש ומתפתח פותר חלק מהמגבלות האלה; נתונים במודל מרובה עננים (multi-cloud) מאוחסנים בתוספת יתירות בשניים או יותר עננים עצמאיים. היתירות מאפשרת למשתמשים לגשת או לשחזר את הנתונים גם כאשר אחד העננים אינו זמין. כמו כן מודל זה מאפשר למשתמשים למקם יותר נתונים ולבצע פעולות קלט ופלט מספקי ענן שמציעים מחיר נמוך יותר או ביצועים גבוהים יותר.

סודיות נתונים

כאשר הנתונים מאוחסנים בענן אחד, הדרך היחידה להבטיח סודיות היא להצפין את הנתונים בצד לקוח לפני העלתו לענן. כך בכל קריאה של הנתונים על המשתמש לפענח את המידע. כמו כן זה מחייב את הלקוח לייצר ולנהל (מקומית או במרוחק) כמויות גדולות של מפתחות הצפנה. הצפנה מבוססת מפתח מבטיחה סודיות חישובית—מונעת מתוקף מוגבל חישובית לגלות את המידע המוצפן. מאחר ופעולת ההצפנה והפענוח נחשבות לפעולות כבדות חישובית, משתמשים רבים עדיין מעלים מידע גלוי לענן.

במודל מרובה עננים, כאשר הנתונים מפוזרים על כמה עננים, אפשר להגן על סודיות המידה בעזרת סכימות חלוקת סוד (secret-sharing schemes). בסכימת חלוקת סוד המידע המקורי של המשתמש מקודד בתוספת מידע אקראי יתיר, המידע המקודד מחולק לחלקים ורק מי שמשיג את כל החלקים (או מספר קבוע מראש של חלקים) יכול לפענח את המידע המקורי. על כן החלקים המקודדים צריכים להישמר במקומות עם אבטחה בלתי תלויה, כפי שנעשה במודל מרובה עננים. חלוקת סוד מבטיחה סודיות מושלמת (information-theoretic security)—גם תוקף בלתי מוגבל חישובית לא יכול לגלות את המידע המקורי. לכן סודיות מושלמת עדיפה על פני סודיות חישובית.

המחקר בוצע בהנחייתם של דר. גלה ידגר, פרופסור איתן יעקובי ופרופסור אסף שוסטר, בפקולטה למדעי המחשב.

תודות

אני רוצה להודות למנחים שלי, ובמיוחד לגלה ידגר, על תמיכתם ועל הזמן הרב שהושקע בי ובמחקר זה. למדתי הרבה מניסיון המחקרי העצום שלהם. היה לי לעונג ולזכות לעבוד איתם וללמוד מהם. בנוסף אני רוצה להודות לאשתי ולהורי על תמיכתם בתקופה מאתגרת זו בחיי.

אני מודה לטכניון על התמיכה הכספית הנדיבה בהשתלמותי.

שילוב יעיל של סודיות וזמינות במערכות אחסון מבוזרות

חיבור על מחקר

לשם מילוי חלקי של הדרישות לקבלת התואר
מגיסטר למדעים במדעי המחשב

רומן שור

הוגש לסנט הטכניון --- מכון טכנולוגי לישראל
אדר ה'תשע"ח חיפה מרץ 2018

שילוב יעיל של סודיות וזמינות במערכות אחסון מבוזרות

רומן שור