The Archistar Secret-Sharing Backup Proxy

Conference Paper · August 2017

DOI: 10.1145/3098954.3104055

3 authors, including:

Andreas Happe
AIT Austrian Institute of Technology
18 PUBLICATIONS **256** CITATIONS
SEE PROFILE

Thomas Lorünser
AIT Austrian Institute of Technology
51 PUBLICATIONS **629** CITATIONS
SEE PROFILE

The Archistar Secret-Sharing Backup Proxy

Andreas Happe
Center for Digital Safety & Security
Austrian Institute of Technology (AIT)
Vienna, Austria
andreas.happe@ait.ac.at

Florian Wohner
Center for Digital Safety & Security
Austrian Institute of Technology (AIT)
Vienna, Austria
florian.wohner.fl@ait.ac.at

Thomas Lorünser
Center for Digital Safety & Security
Austrian Institute of Technology (AIT)
Vienna, Austria
thomas.loruenser@ait.ac.at

ABSTRACT

Cloud-Storage has become part of the standard toolkit for enterprise-grade computing. While being cost- and energy-efficient, cloud storage's availability and data confidentiality can be problematic. A common approach of mitigating those issues are cloud-of-cloud solutions. Another challenge is the integration of such a solution into existing legacy systems. This paper introduces the Archistar Backup Proxy which allows integration of multi-cloud storage into existing legacy enterprise computing landscapes by overloading the industry-standard Amazon S3 protocol. The paper provides multiple lessons-learned during implementation and concludes with a performance evaluation with traditional backup solutions utilizing redundant remote storage.

CCS CONCEPTS

•**Computer systems organization** → **Embedded systems;** *Redundancy;* Robotics; •**Networks** → Network reliability;

KEYWORDS

distributed system, secret sharing, cloud storage, backup system

ACM Reference format:
Andreas Happe, Florian Wohner, and Thomas Lorünser. 2017. The Archistar Secret-Sharing Backup Proxy. In *Proceedings of ARES '17, Reggio Calabria, Italy, August 29-September 01, 2017,* 8 pages.
DOI: 10.1145/3098954.3104055

1 INTRODUCTION

Cloud storage has become a commodity technique, commonly used by companies to dynamically outsource their data storage onto third-party servers. Benefits include increased agility leading to decreased monetary costs, access to managed storage without having to employ storage specialists as well as improved off-site disaster recovery. Drawbacks are the increased dependency upon third-parties, vendor lock-in, loss of data sovereignty and privacy as well as service-level agreements that do not allow contractual enforcement of storage availability and achieved performance.

While companies would like to reap the monetary benefits, data confidentiality issues prevent cloud adaption, esp. with the new, potentially existence-threatening, fines contained in the upcoming General Data Protection Regulation (GDPR) of the European Union. Companies outsourcing their sensitive backup data require strict data confidentiality as well as resilience in case of partial cloud failures.

A new approach that allows to mitigate some of those problems is the *cloud-of-cloud[2]* or *multi-cloud* paradigm. Storage systems that fulfill this technique disperse their data redundantly over multiple independent storage clouds, thus limiting the damage potential of each single storage provider. Furthermore, if secret sharing is used for creating the split-up data, increased data confidentiality can be gained and traditional security assumptions—such as mathematical strength of encryption algorithms—are replaced with a non-collusion assumption between the involved storage providers.

The contribution of this paper is twofold. On the one hand, we will introduce the *Archistar-S3-Proxy* that allows transparent multi-modal data distribution between multiple public clouds. We embrace and enhance existing storage technologies and protocols to allow for integration into existing enterprise storage systems. The basic concept of the Archistar Backup Proxy is: to accept archive data from a backup client or server through the Amazon Simple Storage System (S3) interface, apply secret-sharing to enforce data confidentiality, integrity and availability, and to finally store the encrypted data upon multiple cloud storage providers.

On the other hand, we report significant performance improvements of the *Archistar-Smc* cryptographic library, which was used to build the proxy and is the most versatile and integrated cryptographic secret sharing library available in Java. Archistar-Smc is part of the cryptographic toolkit[22] developed in the PRISMACLOUD project[21] and actively maintained by AIT.

The further layout of this paper is as follows: section 2 gives a rough overview of secret-sharing algorithms and existing multi-cloud storage solutions. Section 3 describes the design choices and high-level architecture of *Archistar-S3-Proxy*; section 4 highlights challenges, limitations and their solutions as well as performance findings discovered during implementation. Section 5 concludes this paper with a short description of our findings as well as teasing contemporary and future work.

2 RELATED WORK

Secret-sharing is the technique utilized by the Archistar Proxy to split up data into multiple fragments. Clients will interact with the proxy through a selected storage protocol. Finally, we will give a typical example of a storage system utilizing the cloud-of-clouds approach.

An important aspect of a software system is the adversary model that the system was designed with. Figure 1a shows a typical cloud backup scenario: multiple backup clients and servers communicate

ARES '17, Reggio Calabria, Italy

978-1-4503-5257-4/17/08…$15.00
DOI: 10.1145/3098954.3104055

with the backup proxy which, in turn, communicates with multiple cloud storage providers at the bottom. A simplified version of the data-flow can be seen in Figure 1b.

In a traditional backup system, everything but the backup client is trusted. The backup servers and storage gateways are fully trusted; the backend storage servers are assumed to behave non-malicious, e.g., stored data might be corrupted due to bitrot but not be maliciously altered or extracted.

The Archistar backup proxy assumes that, in addition to the clients, the backend storage can fail arbitrarily too. Recent cloud outages and breaches[1][2] have shown that this a reasonable assumption. While the Archistar proxy enforces correct behaviour of backup servers, it cannot verify the transmitted plain-text data for correctness. To improve this situation a versioning scheme with the capability to reset to older versions is provided. Archistar assumes the existence of a single proxy; if scaling mandates the usage of multiple proxies, distribution schemes such as Paxos[18] or RAFT[24] can be utilized. If each of those proxies cannot be fully trusted, complex and expensive protocols such as PBFT[6] can be utilized to implement byzantine fault-tolerant systems.

We assume that data-in-transit is protected by state-of-the-art security mechanisms, i.e., TLSv1.2 with data confidentiality and integrity protection. Ideally, for long-term secure storage, the connections would be secured by ITS secure mechanisms like quantum communication[25], however, this technology is very expensive and can only cover limited distances. It is also limited in rate which requires computational secure hybrid solutions [23] to be used in storage scenarios anyways.

2.1 Secret-Sharing

Secret-sharing describes a family of algorithms that allow to split up plain-text data into n parts, k of which are needed to reconstruct the original data. This definition implies that $k < n$; if $k \leq n$ then $n - k$ missing parts can be tolerated during reconstruction. The archetypal secret-sharing algorithm is Shamir's Secret Sharing[28]. While introducing a high dependency upon generated random numbers as well as significant storage overhead this algorithm is information theoretical secure, i.e., even an adversary with unlimited processing power can only randomly guess the plain-text data. Another often used algorithm is Rabin[26] which increased information safety by adding redundancy during data distribution. While not improving confidentiality it allows for highly efficient data dispersal. Krawczyk combined both schemes in [16]. Plain-text data is symmetrically encrypted using a generated secret-key. The encrypted data is dispersed using Rabin while the secret key is distributed using Shamir's Secret Sharing. While this only provides computational security, this approach yields high storage efficiency. Table 1 gives an overview of the discussed algorithms and their characteristics.

2.2 Storage Protocols

Storage systems can be classified according to the data access methods they are providing to their clients. Most storage interfaces fall into one of the following classes: remote file systems, block storage or key-value stores[3]. In addition systems can be classified due to the CAP theorem[9]—denoting Consistency, Availability and Partition Tolerance. The theorem states, that a system can fulfill a maximum of two out of three of those parameters.

Remote file systems provide hierarchical data storage with high semantics[27]. In case of cluster file systems parallel write-operations by multiple clients while keeping strong consistency are commonly supported. This high feature level is bought by complexity.

Block-level storage offers a virtual block device, e.g., a distributed hard-drive. Data representation is not files, but fixed-size storage blocks. It is the client's responsibility to provide a file system with all inherent complexities on top of the virtual block device. This reduced feature allows for increased simplicity and thus low latency. In addition the block-level storage is well suited for data deduplication on the lowest layer. This improves storage efficiency but has the inherent drawback of reduced data safety due to reduced redundancy as well as a potential negative confidentiality impact[13].

Key-Value stores provide functionality akin to non-hierarchical dictionaries[11]. Similar to block-based storage systems they utilize unique keys for identifying data, in contrast to block-storage they allow storage of arbitrarily sized values. They focus upon availability and horizontal scale-out, i.e., sharding, and have become prominent for cloud applications. The foremost known key-value network protocol is the industry-standard *Amazon S3* protocol which is served over HTTPS.

Table 1: Characteristics of selected secret-sharing algorithms. Storage overhead is noted as factor based upon the original storage size l. n is the total amount of servers needed, k denotes how many fragments are needed to reconstruct the original data. An overhead factor of 1.0 denotes that the resulting share size is the same as the original data, i.e., that there is no overhead.

Algorithm	Confidentiality	Storage Overhead
Shamir	ITS	n
Rabin		n/k
Krawczyk	computational	$n/k + size_{key} * n/l$

2.3 Multi-Cloud Storage

Fault-tolerant cloud storage is a well researched area[30]. Common distinctive features range from concurrent multi-client support, concurrent updates, used consistency model, data deduplication, fault tolerance and cloud storage provider requirements.

RACS[1] utilizes erasure-coding to distribute data over multiple storage clouds. Their main goal is to prevent vendor lock-in and to achieve high availability; privacy concerns are not discussed. It mimics the Amazon S3 interface for communication with its clients. If a single RACS installation becomes a performance bottleneck, distributed RACS can be deployed: it uses Apache Zookeeper[15]

[1] Gitlab data loss: https://about.gitlab.com/2017/02/01/gitlab-dot-com-database-incident, accessed 06/12/2017.
[2] Amazon S3 outage: https://aws.amazon.com/message/41926/

[3] Please note that we are focusing upon file-based storage systems and are not analyzing structured storage systems such as relational databases.

for message ordering/storage and is thus a crash-fault tolerant solution.

The HAIL system[5] focuses upon high-availability and integrity protection within the cloud; data privacy is not of primary concern. To achieve high availability, data is distributed (using erasure codes) upon multiple clouds. Data stored upon a single server is also redundantly stored to increase its resistance against bitrot. To verify the availability and correctness of data, a proof-of-retrievability protocol based upon active servers has been developed.

DepSky[4] offers an object-store interface on top of passive storage clouds. Its data objects utilize cryptographic hashes for integrity control; short-time version numbers provide for concurrent updates. As no active server components can be used, the system cannot cope with malicious writers. Multiple concurrent writers are supported through client-side locks: this allows for obstruction-free, but not wait-free, operation. Cloud providers are allowed to fail in Byzantine ways. Confidentiality is optionally supported by secret-sharing techniques in the DepSky-CA variant.

Fork-based systems[19] achieve consistency by exploiting version information. Multiple clients update the server-side data and, through that, create potential conflicting change histories. If a malicious change has been detected, clients are responsible to rollback to a known good version. This approach is well suited if clients can detect failures, are able to perform the rollback operation and incorporating potential malicious data can be coped with.

Another potential solution is the combination of a byzantine fault-tolerant distribution algorithm with erasure-coding[10]. Byzantine faults are arbitrary faults, well suited to model, e.g., a malicious attacker as well as bit-rot due to erroneous hardware. A drawback of PBFT-based solutions is the dependency upon active servers, the need for servers communicating with each other as well as the high message count and round-trip overhead. Using erase-codes does not provide for data confidentiality; a natural evolution of this approach is the addition of secret-sharing[20]. The adversary model for such a system becomes complex, esp. if malicious clients should be able to be coped with[12].

3 ARCHITECTURE

The Archistar Backup Proxy uses secret-sharing to split up archive data—provided by an existing backup solution—into encrypted shares and then distributes those shares upon multiple potentially untrusted cloud storage providers. A simplified representation of our system architecture is shown in Figure 1a, a simplified version of the dataflow is shown in Figure 1b.

We target the enterprise backup sector, this leads to multiple simplifications for the resulting software architecture. Enterprise backup clients typically connect to a single backup server which used to store the clients' data upon tape drives but recently have begun to exchange those tape decks with cheap online cloud storage. Existing Enterprise Backup Solutions commonly support the Amazon S3 protocol for backend storage. By providing an emulation of this protocol, the Archistar Backup Proxy becomes a drop-in network component. While this improves the interoperability, the situation is not perfect: not all backup solutions utilize the same Amazon S3 subset.

The resulting system is inherently single-user: while multiple clients connect to the backup server, just a single backup server will connect to the Archistar Backup Proxy. While the backup proxy itself might provide multi-user support to its clients, the overall system is still a single-client system with the backup proxy being the single storage client. The single-user nature of the system allows to delegate large parts of the user authentication and authorization problem to the backup server software. As we target real-world deployments, we cannot assume active servers with code-execution capabilities. This reduces our server-side options as many storage systems mandate active servers with communication channels between them, e.g., PBFT-based solutions[10], or active servers with user-supplied code, e.g., fork-consistency based systems.

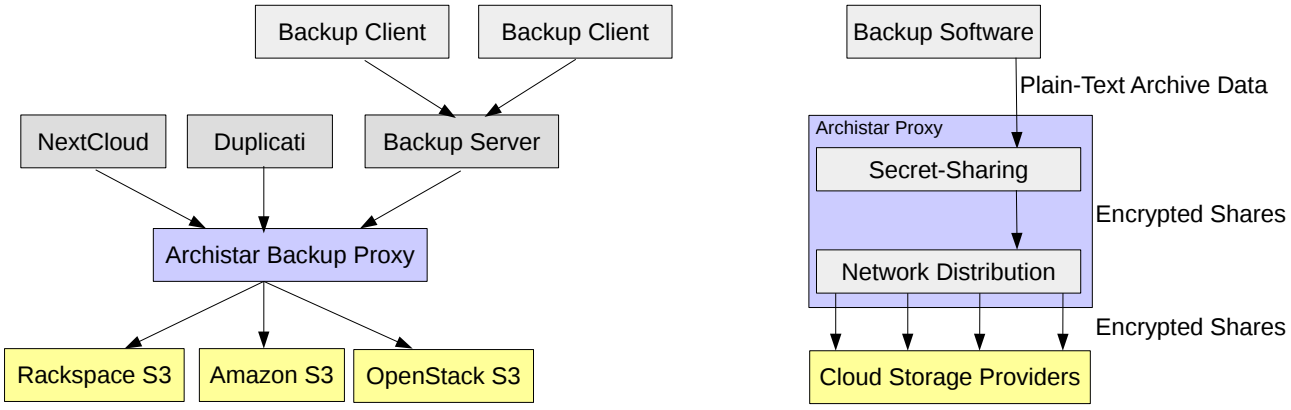
A high-level benefit of the backup storage sector is its focus upon bandwidth. In contrast, desktop-level interactive applications are forced to focus upon latency. As we can thus allow for higher latencies, this enables easy implementation of batching and caching, thus further improving the achieved bandwidth.

As mentioned, we have chosen the Amazon S3 protocol for client interactions due to its mass market adaption. We also support the Amazon S3 protocol for interaction with our backend storage providers but offer additional in-memory and local-storage providers for testing purposes. The concrete storage driver implementations are abstracted behind an interface, so supporting other cloud providers or modes of storage is a matter of implementing few very basic methods. Apart from utility functions — for connecting and disconnecting, returning status information — one only needs to provide methods for storing, retrieving, and deleting binary objects. As the proxy server keeps its own metadata within its index, no methods for listing directories or retrieving file stats are needed.

The index itself is a container for file data, i.e., a file's original name, size, SHA-256 and MD5[4] hashes, modification date, content type, used secret-sharing algorithm and metadata. In addition, file data includes information on where its secret-shared parts are stored. This is needed, as we store each under a separate identifier to hamper data analysis. Apart from that, each index can — depending on the versioning configuration — contain a reference to its predecessor. As each index includes information about all available files and directories, this allows for implicit versioning of all data. To reduce storage overhead, the number of stored versions, reaching from zero to infinity, can be configured through the backup server's configuration.

Secret-sharing is utilized to distribute the index upon the available storage clouds. It is therefore only identified by its unique random identifier[5]; to identify the current index, a special file with a fixed name is used akin to a super-block in filesystem design. In order not to have to retrieve the current index on every operation and thus spare round-trips, it is cached locally. Storage operations that do not manipulate actual file data, are thus "cheap". If we assume single-proxy operation, we do not have to check if our locally cached index and the server-side index are identical. Listing all stored files, or retrieving the metadata of one specific file, does not entail any backend operations. This is paramount for performance,

[4]The MD5 hash is needed for compatibility with the Amazon S3 protocol.
[5]Other than regulars files though, all secret-shared parts of the index have the same identifier upon all used storage providers.



(a) High-level network system overview.

(b) Example data-flow from backup client.

Figure 1: High-Level network system overview and example data-flow from backup clients to the cloud storage servers.

as some clients issue a surprisingly high number of such requests before and after each actual file operation. In addition this allows the backup server to better cope with the potential high-latency during accessing networked Amazon S3-compatible storage servers.

3.1 Theoretical Secret-Sharing Performance

The performance overhead of the Archistar Proxy system can be measured through latency, minimal number of needed servers, storage overhead and throughput. Given a storage system that must be able to cope with f faulty storage locations, traditional cloud storage systems that provide redundant fault-tolerant storage must distribute the original data upon (at least) $k = f + 1$ servers. The storage overhead is thus $f + 1$.

Erasure-Coding has a similar amount of needed servers, but the storage-overhead per server is reduced by a factor of k.

With information-theoretical secure secret-sharing there is the additional condition that collaborating storage providers should not be able to reconstruct the original data: $k = f + 1$, $n - k > f$. If we transform the latter this leads to $n = 2f + 1$. Table 2 shows the overhead results for different values of f. It can be seen that while secret-sharing increases the amount of minimal servers, the overall storage overhead depends upon the chosen secret-sharing algorithm. When information-theoretical secret-sharing is used, the overhead is higher than a comparable redundant backup solution. When computational secure secret-sharing is used, it is lower.

Another important metric for backup systems is their throughput. The achieved (simplified) throughput is the minimum of the incoming (local) network throughput, the secret-sharing engine's throughput as well the achievable outgoing (public) network throughput. We assume the internal network to be of infinite capacity and that our baseline — a traditional backup solution that uses simple redundancy — saturates the available outgoing network bandwidth. If we assume a scenario of $f = 1$, i.e., the user wants at least minimal safety from redundancy, then the impact upon throughput should be the ratio of secret-sharing overhead divided by the traditional overhead. When using the estimated overhead from Table 2, it can be seen that information-theoretical secure secret-sharing would decrease the throughput, while computational secure secret-sharing would actually improve it. The throughput of the used secret-sharing engine thus becomes the limiting factor. As a reference, Table 4 shows the measured performance of our secret-sharing library upon multiple platforms.

3.2 Availability Model

In reliability theory an n-component system that works if and only if at least k of the n components work is called a k-out-of-n:G system. The presented storage system is exactly implementing such a structure in a multi-cloud setting which lets us directly apply some results from reliability theory in our availability analysis.

In particular, the reliability $R(k, n)$ of a k-out-of-n:G system with i.i.d. components, i.e., components which are independent of each other, is equal to the probability that the number of working components is greater than or equal to k. In particular the reliability is calculated as follows.

$$R(k,n) = \sum_{i=k}^{n} \binom{n}{i} p^i q^{n-1} \tag{1}$$

The k-out-of-n is a generic model for adding fault tolerance to systems by increasing redundancy, which is exactly what we are doing with secret sharing in the Archistar system, if we leave the security aspects aside for this treatment.

If we compare the different settings presented in the previous section with the reliability model we get the following results. For the case of data replication we have $k = 1$, which leads to the analogous of a parallel system in the reliability model and $R(1, n) = 1 - \prod_{i=1}^{n}(1 - p_i) = 1 - (1 - p)^n$. For the cases of perfectly secure (ITS) secret sharing and computational secret sharing the reliability parameters can flexibly be adjusted through encoding between $1 \leq k \leq n$, which leads to a non trivial k-out-of-n system if k is selected accordingly ($k > 1$ and $k < n$). However, if the redundancy is fully removed for security reasons ($k = n$), the systems becomes a simple series system with $R(n, n) = \prod_{i=1}^{n} p_i = p^n$. Thus, from a reliability standpoint, both secret sharing variants provide the same

Table 2: Comparison of minimal server amount and storage overhead between different storage schemes. The baseline for the overhead is given as a factor based upon the original data amount.

Faulty Servers	Server Amount		Storage Overhead		
	Replication	Secret-Sharing	Traditional Replication	Computational Secret-Sharing	Perfect (ITS) Secret-Sharing
$f = 1$	$n = k = 2$	$k = 2, n = 3$	2	1.5	3
$f = 2$	$n = k = 3$	$k = 3, n = 5$	3	1.66	5
$f = 3$	$n = k = 4$	$k = 4, n = 7$	3	1.75	7

level of reliability, although providing different levels of security and storage overhead.

Now, if we use the previous treatment to model the availability of our solution, the basic characteristics can be directly applied. As we show here, we can also use this approach to design our system based on availability criteria we have to fulfill for the data stored. Enabling this SLA tailoring via multi-cloud configurations is very attractive, because the standard cloud storage market provides only limited flexibility in the configuration of service level agreements (SLA). In many cases main design criteria like availability goals are not even clearly stated in provider SLA[6], nor is there any real compensation foreseen in case of violation, except for some minor service credits. In fact, serving standardized SLAs to customers is a major feature of cloud computing which helps to enable the elasticity and self-service capabilities the customers want to have. However, as a downside of this approach, it is very difficult for customers to find offerings which perfectly fit their particular needs and almost impossible to negotiate special conditions.

Archistar is trying to solve this problem in a different way, i.e., by letting the customer design a storage system according their needs and requirements as a fault tolerant composition of different cloud offerings. In particular, the k-out-of-n paradigm is used to design systems which can theoretically provide arbitrary high levels of availability. Availability classes are typically given as number of leading nines of the availability value, i.e., a "three nines" availability means 99.9% which corresponds to a downtime of $8.76h$ per year or $43.8min$ per month. We used this type of availability classes to demonstrate the theoretical values we can reach in our system with reasonable number of storage nodes.

In table 3 we show the calculated availability classes for different configurations of n and k, whereby an overall availability of 98% ($p = 0.98$) is assumed for the individual cloud storage offerings used to store the data fragments. It is easy to see, that for all typical requirements configuration parameters exist. The system is giving the cloud customer much more flexibility and enables him to design his own SLA for a virtual Archistar storage service with respect to availability, confidentiality and integrity on top of existing cloud offerings. This can also help to speed up and improve the cloud migration process in general[14]. Furthermore, if the configurations would be matched against cloud services databases, the best provider offerings can be selected to also get a price optimal solution. Nevertheless, a more detailed model also considering failure modes like network outages and data loss would be desirable and is left for future work.

[6]e.g. see Amazon S3 service: https://aws.amazon.com/s3/sla/ (accessed 6/17/2017)

Table 3: Number of leading nines for system reliability $R(k, n)$ for given n and k and a individual storage node reliability of $p = 0.98$, which is a typical value taken from cloud storage provider SLA.

n	k											
	1	2	3	4	5	6	7	8	9	10	11	12
3	5	3	1	0	-	-	-	-	-	-	-	-
4	7	5	3	1	0	-	-	-	-	-	-	-
5	8	6	4	2	1	0	-	-	-	-	-	-
6	10	8	6	4	2	1	0	-	-	-	-	-
7	12	9	7	5	4	2	1	0	-	-	-	-
8	14	11	9	7	5	3	2	1	0	-	-	-
9	15	13	10	8	6	5	3	2	1	0	-	-
10	17	14	12	10	8	6	5	3	2	1	0	-
11	19	16	14	11	9	8	6	4	3	2	1	0
12	20	18	15	13	11	9	7	6	4	3	2	1
13	22	19	17	15	12	11	9	7	5	4	3	2
14	24	21	18	16	14	12	10	8	7	5	4	3
15	25	23	20	18	16	14	12	10	8	7	5	4

3.3 Backend Concerns

One of our backup proxy's goals is interoperability. Thus we not just provide a standard S3 interface to our clients, but also must allow for multiple backend storage options. To achieve this, our design is based on the lowest sensible common denominator. For the most part, this is not a significant limitation—our requirements are modest, and major cloud providers do or will soon provide a matching set of features. One notable exception is Amazon's provided data consistency model for its operations. Its eventual-consistency model is far weaker than the strong-consistency model provided by other cloud providers. As a consequence, we can not support parallel data operations by multiple proxies on shared sets of stored data. A single proxy installation is therefore intended to only ever work on a set of stored data that does not overlap with that of any other proxy.

A good example for an advanced but well-supported utilized feature is metadata management. Almost all cloud storage providers provide (under various names) operations on user-defined metadata that can be manipulated separately from the object to which they are attached. Though the concrete limitations regarding size and shape differ from provider to provider, our requirements in this regard are modest enough that all the file-level metadata needed for our secret sharing engine can be stored in that way.

3.4 Client-Facing Concerns

The Archistar backup proxy provides a subset of the industry-standard Amazon S3 protocol. The Amazon S3 protocol is a RESTful HTTP-based key-value storage protocol. Data is identified by non-hierarchical unique alpha-numeric keys; data itself is assumed to be a binary large object. To better implement a filesystem-like interface, keys can be grouped by a common prefix — this allows to emulate directories by using a full path as key. Each object can contain both predefined, e.g., user and access information, and user-defined meta-data.

To allow upload of large files through the HTTP a multi-part upload functionality is provided. This introduces a transaction-like process, within which a client can upload multiple smaller parts of a file and where during the finalization of the transaction, those parts are combined to the finally stored object.

We implement basic authentication using statically configured credentials against which incoming client requests are authenticated. We were not able to implement the sophisticated authorization features of S3 as this would require either deep access to Amazon customer data, or replicating its *Identity and Access Management (Amazon IAM)* infrastructure. As this is a common problem with Amazon S3-compatible implementations, this lack is not serious — all of the backup servers/clients which we have so far integrated, utilized only a small part of the S3 interface.

A benefit of using the S3 protocol is that, due to it being a key/value store, deduplication is relatively easy[29]. Deduplication is an important feature as many backup clients perform UNIX-style atomic file updates/uploads by initially uploading a temporary file, then making a copy of it under the final name, and finally deleting the temporary file. Deduplicating here has the obvious benefit that only the initial upload file operation is actually being performed on a data-level, while the subsequent operations only modify data within the — potentially cached — index. This reduces round-trips and thus improves performance.

One feature that the S3 protocol offers and will be supported by our backup proxy is versioning. Already it is possible to configure the proxy so that files are never actually deleted, or only after certain criteria are met. The open question is whether, and if so, how to expose this functionality via S3. The straightforward solution would of course be to integrate it with the versioning facilities of S3, but not many client actually use this feature. For most clients, S3 is just one of many supported backends; they therefore avoid depending too much on S3-specific features, or even on features that are specific to cloud key/value stores.

4 IMPLEMENTATION AND EVALUATION

This section describes findings that were discovered during implementation. In contrast to the "Architecture"-findings, these findings are more concerned with mechanics and protocol choices.

4.1 S3 Integration Challenges

During implementation of the proxy prototype, we discovered significant negative performance interactions between our initial data representation and the Amazon S3 protocol, leading to decreased performance and scalability.

As mentioned before, S3 is a key/value store with an assumed flat, i.e., non hierarchical, key space whose internal value structure is orthogonal to the protocol. Sadly, this is not what clients expect: when dealing with files, instead of a flat name space, one would probably like to have directories. For handling large files, incremental up- and download facilities would be useful. S3 provides for this: in the case of directories, this is done by treating directory names as "common prefixes". For incremental uploads of large files, there is a special "Multipart Upload" operation. As for downloading big files, S3 makes use of byte-wise access through HTTP Range headers[7]. As our security model assumes that we only serve (and integrity-check) whole files, this introduces a payoff between security and performance — either we retrieve, reconstruct, and serve just the requested part of the file but are then not able to verify its integrity, or we can retrieve the whole file, check it, and serve only the requested part. This is further exacerbated by the fact that client can — via the Range headers — specify arbitrary ranges to be retrieved. The size of the "chunks" in which the file is retrieved cannot a priori be determined and thus, no check-sum be pre-calculated. A 150MB file might be retrieved in 2 chunks, or in 16.

This does not only impact performance. Our current prototype implementation is based on Java byte arrays for incoming data, processing as well as for outgoing data. This has obvious negative implications for memory usage, but also less obvious implications for stability. For every l bytes of input, at least $n + 1$ byte arrays of length l have to be allocated by the Java virtual machine. In typical use, allocations do not fail, but in principle, the JVM can throw an OutOfMemoryError at any time — as we have found, frequently accessing files of moderate size such as 50 Megabyte are sometimes enough to trigger this condition. This happens exactly when a client sends multiple requests for different ranges of the same file.

4.2 Secret-Sharing Engine Performance

When analyzing the sequential single-thread performance of our data-processing engine, the typical culprits can be found: finite field multiplication and random number generation. Speeding up finite field multiplication is an intensively researched subject, but for our specific purposes of bit fields of width 8, we found the approach of using lookup tables more than adequate. A high random number generation rate is paramount, because when using Shamir's secret-sharing algorithm, for every byte of input, $k-1$ bytes of randomness must be generated. The availability of a high-performance hardware random number generator is thus highly recommended. The single-thread performance of our secret-engine can be seen in Table 4.

Our current prototype uses a sequential and single-threaded secret-sharing engine. To improve performance we are offering a first version of a parallelized erasure coding algorithm. We are investigating the potential of splitting up the workload upon multiple CPU cores, esp., to be better suited for operation on dedicated network hardware which often provides many low-performance CPU cores. Rabin's algorithm allows for easy optimization, as each of the n output shares can be generated on a different core because no synchronization between them is necessary. Shamir's algorithm depends upon $k-1$ generated random bytes for every byte of input,

[7] Specified in RFC 2616 §14.35, current version: RFC 7233.

Table 4: Single-Core throughput upon plaintext-data of different secret-sharing algorithms on different Intel CPUs with $n = 4, k = 3$, measurements in kilobyte/second. The Intel i5 is a good example of a middle-range Desktop CPU while the Atom C2758 is typical for an embedded high-range CPU.

CPU	Speed	Algorithm	Split-Up	Combine
i5-4690K	3.5GHz	Shamir	42533.7	54179.9
C2758	2.4Ghz	Shamir	9212.8	10790.3
Exynos A9	1.7GHz	Shamir	3400.6	3940.7
ARM A53	1.2Ghz	Shamir	1695.1	2034.9
i5-4690K	3.5GHz	Rabin	123746.2	129211.4
C2758	2.4Ghz	Rabin	28603.4	26056.0
Exynos A9	1.7GHz	Rabin	9915.3	10559.4
ARM A53	1.2Ghz	Rabin	5102.1	5346.6
i5-4690K	3.5GHz	Krawczyk	79534.0	80313.7
C2758	2.4GHz	Krawczyk	17754.7	16828.3
Exynos A9	1.7GHz	Krawczyk	6626.8	6813.0
ARM A53	1.2Ghz	Krawczyk	3591.4	3773.0

so a parallel implementation would either have to pre-generate the extra coefficients, or synchronize their generation across multiple threads. Preliminary performance numbers for a parallelized Rabin can be seen in Table 5. Please note, that we are currently parallelizing up to n (maximum number of shares) threads — in our test-case, the embedded Intel Atom processor would offer more cores than our maximum number of shares, thus no full utilization of the processor cores is achieved. There is a slight performance hit when the multi-threaded code is run on a single-core processor: this is grounded within the initial setup overhead of the multi-threaded code. To get the best performance, two separate versions of the library (or instantiations within the library) for single- and multi-core usecases might be advantageous.

The speed-up for Rabin on a quad-core was around the expected factor of four. Krawczyk is a combination of Shamir, Rabin and symmetrical encryption. The secret-sharing key is shared using Shamir, the original data is encrypted using a traditional symmetric encryption and then shared with Rabin. If the existing symmetric encryption library (BouncyCastle in our case) scales as well as our new parallelized Rabin code, then Krawczyk should also improve around a factor of 4. As our empirical tests have shown, a factor of 2 is achieved. This indicates that the existing symmetric encryption engine also has become a bottleneck with our new code. Future research into alternative cryptographic engine as well as into Java 9 (which improves hardware support for cryptographic directives prevalent in modern processors) is planned.

Another possibility is to split the input across its length. In that case, the problem becomes the assembly and ordering of the generated output. Currently, this is feasible as both input and output buffers are byte arrays. Parallelization can be achieved by computing indices within those arrays. Alas, as mentioned before, byte arrays are not a scalable solution. Therefore, we will switch to a stream-based implementation, which renders this parallelization approach infeasible.

Table 5: Preliminary performance (Version *a8a344b*) with multi-core enabled Rabin Data Dispersal. As Rabin is also used by Krawczyk, its performance should also improve. Please note, that currently a maximum number of n threads is used. The tested Atom C2758 board offers 8 cores, so it is not utilized to the maximum of its abilities.

CPU	Speed	Algorithm	Split-Up	Combine
i5-4690K	3.5GHz	Shamir	53753.3	52378.0
C2758	2.4Ghz	Shamir	8192.0	7881.5
Exynos A9	1.7GHz	Shamir	6917.8	3932.0
ARM A53	1.2Ghz	Shamir	3502.4	2056.9
i5-4690K	3.5GHz	Rabin	694237.3	124498.5
C2758	2.4Ghz	Rabin	99417.5	26072.6
Exynos A9	1.7GHz	Rabin	67590.8	10529.6
ARM A53	1.2Ghz	Rabin	40634.9	4395.2
i5-4690K	3.5GHz	Krawczyk	167868.9	79844.1
C2758	2.4GHz	Krawczyk	32507.9	16862.9
Exynos A9	1.7GHz	Krawczyk	16430.0	6949.4
ARM A53	1.2Ghz	Krawczyk	9669.5	3765.1

Table 6: Impact of Processor Architecture and difference incoming data sizes upon Rabin throughput. Multi-threaded algorithm used, all Values in kByte per Second.

Data Size	Intel Core i5-4570		ARM A9	
	Split-Up	Combine	Split-Up	Combine
4 kB	117028.6	84553.6	39309.0	9992.7
64 kb	481882.4	121904.8	74744.5	10890.7
128 kb	460224.7	123373.5	76560.7	10922.7
512 kb	630153.8	124121.2	65958.1	10873.4
1024 kb	650158.7	123746.2	107789.5	10870.5
2048 kb	660645.2	124498.5	111304.3	10884.9
4096 kb	620606.1	124878.0	111404.3	10873.4

For further optimization, we performed pre-processing of constant factors for common operations. E.g., when sharing a secret, for every generated output share, all multiplications are performed with the same factor. We are currently investigating another example of preprocessing: when reconstructing a secret, a decoder matrix has to be generated — instead of using generic matrix multiplication we could decompose and simplify the matrix.

When testing multiple input file sizes, we found that a block size of 4 Kilobyte produced significantly reduced performance. With larger input sizes, the performance was generally constant on the Intel Core i5 CPU. On passively cooled embedded devices (ARM as well as Intel Atom processors) the power- and thermal management prevented the creation of reproducible performance benchmarks. Future research in to the power consumptions and thermal impact of differnet algorithms might be needed. An overview of the recorded throughput values can be seen in Table 6.

5 FUTURE WORK AND CONCLUSION

We presented the Archistar Backup Proxy that allows integration of secret-sharing multi-clouds into existing legacy enterprise storage systems. We addressed the common belief that secret-sharing techniques are problematic for both storage overhead and throughput.

In future work, we hope to increase the achieved bandwidth through the mentioned secret-sharing engine improvements. Another future work-item is a detailed performance comparison with competing solutions such as DepSky[4] and byzantine-fault tolerant active solutions. Feature-wise, first attempts of integrating private information retrieval (PIR[7]) and remote data checking (RDC[3]) are underway.

Furthermore we will integrate novel verification protocols[8, 17] to increase the systems resilience and security. The protocols have been specifically designed for Archistar and with with efficiency in mind, i.e., for large distributed storage of unstructured data.

ACKNOWLEDGMENTS

This work was in part funded by the European Commission through grant agreement no 644962 (PRISMACLOUD).

REFERENCES

[1] Hussam Abu-Libdeh, Lonnie Princehouse, and Hakim Weatherspoon. 2010. RACS: A Case for Cloud Storage Diversity. In *Proceedings of the 1st ACM Symposium on Cloud Computing (SoCC '10)*. ACM, New York, NY, USA, 229–240. DOI: http://dx.doi.org/10.1145/1807128.1807165

[2] Mohammed A AlZain, Eric Pardede, Ben Soh, and James A Thom. 2012. Cloud computing security: from single to multi-clouds. In *System Science (HICSS), 2012 45th Hawaii International Conference on*. IEEE, 5490–5499.

[3] Giuseppe Ateniese, Randal Burns, Reza Curtmola, Joseph Herring, Osama Khan, Lea Kissner, Zachary Peterson, and Dawn Song. 2011. Remote data checking using provable data possession. *ACM Transactions on Information and System Security (TISSEC)* 14, 1 (2011), 12.

[4] Alysson Bessani, Miguel Correia, Bruno Quaresma, Fernando André, and Paulo Sousa. 2013. DepSky: dependable and secure storage in a cloud-of-clouds. *ACM Transactions on Storage (TOS)* 9, 4 (2013), 12.

[5] Kevin D Bowers, Ari Juels, and Alina Oprea. 2009. HAIL: A high-availability and integrity layer for cloud storage. In *Proceedings of the 16th ACM conference on Computer and communications security*. ACM, 187–198.

[6] Miguel Castro, Barbara Liskov, and others. 1999. Practical Byzantine fault tolerance. In *OSDI*, Vol. 99. 173–186.

[7] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. 1995. Private information retrieval. In *Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on*. IEEE, 41–50.

[8] Denise Demirel, Stephan Krenn, Thomas Lorünser, and Giulia Traverso. 2016. Efficient and Privacy Preserving Third Party Auditing for a Distributed Storage System. In *11th International Conference on Availability, Reliability and Security, {ARES} 2016, Salzburg, Austria, August 31 - September 2, 2016*. {IEEE} Computer Society, 88–97. DOI: http://dx.doi.org/10.1109/ARES.2016.88

[9] Seth Gilbert and Nancy Lynch. 2002. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *Acm Sigact News* 33, 2 (2002), 51–59.

[10] Garth R Goodson, Jay J Wylie, Gregory R Ganger, and Michael K Reiter. 2004. Efficient Byzantine-tolerant erasure-coded storage. In *Dependable Systems and Networks, 2004 International Conference on*. IEEE, 135–144.

[11] Jing Han, E Haihong, Guan Le, and Jian Du. 2011. Survey on NoSQL database. In *Pervasive computing and applications (ICPCA), 2011 6th international conference on*. IEEE, 363–366.

[12] Andreas Happe, Stephan Krenn, and Thomas Loruenser. 2016. Malicious Clients in Distributed Secret Sharing Based Storage NetworksQufh. (2016).

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

[14] Aleksandar Hudic, Matthias Flittner, Thomas Lorünser, Philipp M Radl, and Roland Bless. 2016. Towards a Unified Secure Cloud Service Development and Deployment Life-Cycle. In *11th International Conference on Availability, Reliability and Security, {ARES} 2016, Salzburg, Austria, August 31 - September 2, 2016*. {IEEE} Computer Society, 428–436. DOI: http://dx.doi.org/10.1109/ARES.2016.73

[15] Patrick Hunt, Mahadev Konar, Flavio Paiva Junqueira, and Benjamin Reed. 2010. ZooKeeper: Wait-free Coordination for Internet-scale Systems.. In *USENIX annual technical conference*, Vol. 8. 9.

[16] Hugo Krawczyk. 1993. Secret sharing made short. In *Annual International Cryptology Conference*. Springer, 136–146.

[17] Stephan Krenn, Thomas Loruenser, and Christoph Striecks. 2017. Batch-verifiable Secret Sharing with Unconditional Privacy. In *Proceedings of the 3rd International Conference on Information Systems Security and Privacy - Volume 1: ICISSP,*. INSTICC, ScitePress, 303–311. DOI: http://dx.doi.org/10.5220/0006133003030311

[18] Leslie Lamport and others. 2001. Paxos made simple. *ACM Sigact News* 32, 4 (2001), 18–25.

[19] Jinyuan Li, Maxwell N Krohn, David Mazières, and Dennis Shasha. 2004. Secure Untrusted Data Repository (SUNDR).. In *OSDI*, Vol. 4. 9–9.

[20] Thomas Loruenser, Andreas Happe, and Daniel Slamanig. 2015. ARCHISTAR: towards secure and robust cloud based data sharing. In *Cloud Computing Technology and Science (CloudCom), 2015 IEEE 7th International Conference on*. IEEE, 371–378.

[21] Thomas Lorünser, CharlesBastos Rodriguez, Denise Demirel, Simone Fischer-Hübner, Thomas Groß, Thomas Länger, Mathieu des Noes, HenrichC. Pöhls, Boris Rozenberg, and Daniel Slamanig. 2015. Towards a New Paradigm for Privacy and Security in Cloud Services. In *Cyber Security and Privacy*, Frances Cleary and Massimo Felici (Eds.). Communications in Computer and Information Science, Vol. 530. Springer International Publishing, 14–25. DOI: http://dx.doi.org/10.1007/978-3-319-25360-2_2

[22] Thomas Lorünser, Daniel Slamanig, Thomas Länger, and Henrich C Pöhls. 2016. PRISMACLOUD Tools: A Cryptographic Toolbox for Increasing Security in Cloud Services. In *11th International Conference on Availability, Reliability and Security, {ARES} 2016, Salzburg, Austria, August 31 - September 2, 2016*. {IEEE} Computer Society, 733–741. DOI: http://dx.doi.org/10.1109/ARES.2016.62

[23] Andreas Neppach, Christian Pfaffel-Janser, Ilse Wimberger, Thomas Loruenser, Michael Meyenburg, Alexander Szekely, and Johannes Wolkerstorfer. 2008. Key management of quantum generated keys in IPSEC. In *International Conference on Security and Cryptography SECRYPT 2008 July 26 2008 July 29 2008 ({SECRYPT} 2008 - International Conference on Security and Cryptography, Proceedings)*. Inst. for Syst. and Technol. of Inf. Control and Commun., 177–183. http://www.bibsonomy.org/bibtex/24935e22403086f8533982907489e1a0f/dblp

[24] Diego Ongaro and John K Ousterhout. 2014. In Search of an Understandable Consensus Algorithm.. In *USENIX Annual Technical Conference*. 305–319.

[25] Momtchil Peev, Thomas Langer, Thomas Lorunser, Andreas Happe, Oliver Maurhart, Andreas Poppe, and Thomas Themel. 2009. The SECOQC Quantum-Key-Distribution network in Vienna. In *Optical Fiber Communication - incudes post deadline papers, 2009. OFC 2009. Conference on*. 1–3.

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

[27] Mahadev Satyanarayanan. 1990. A survey of distributed file systems. *Annual Review of Computer Science* 4, 1 (1990), 73–104.

[28] Adi Shamir. 1979. How to share a secret. *Commun. ACM* 22, 11 (1979), 612–613.

[29] Youngjoo Shin, Dongyoung Koo, and Junbeom Hur. 2017. A Survey of Secure Data Deduplication Schemes for Cloud Storage Systems. *ACM Comput. Surv.* 49, 4, Article 74 (Jan. 2017), 38 pages. DOI: http://dx.doi.org/10.1145/3017428

[30] Minqi Zhou, Rong Zhang, Wei Xie, Weining Qian, and Aoying Zhou. 2010. Security and privacy in cloud computing: A survey. In *Semantics Knowledge and Grid (SKG), 2010 Sixth International Conference on*. IEEE, 105–112.