ARCHISTAR: Towards Secure and Robust Cloud Based Data Sharing

Thomas Loruenser, Andreas Happe
Department Digital Safety & Security
AIT Austrian Institute of Technology GmbH
Vienna, Austria
{thomas.loruenser|andreas.happe}@ait.ac.at

Daniel Slamanig
Graz University of Technology
Graz, Austria
daniel.slamanig@iaik.tugraz.at

***Abstract*—Cloud based collaboration gives rise to many new applications and business opportunities in both the private and the business domain. However, building such systems in a secure and robust manner is a challenging task. In this paper, we present a new architecture for secure cloud based data sharing called ARCHISTAR. It builds upon a distributed storage system and thus avoids any single point of trust or failure. Besides providing confidentiality of data, our focus is on availability and in particular on robustness against active attacks or failures. Our system provides full multi-user support and enables advanced sharing scenarios without complex key management and revocation mechanisms. We also present a prototype implementation of the ARCHISTAR system and discuss open issues.**

***Keywords*-cloud security; cryptography; distributed systems; information sharing;**

I. Introduction

Since its emergence around 2007, cloud computing has become an important part within the IT infrastructure and service provisioning domain. One of it's main pillars is data storage. Due to the ubiquitous accessibility of cloud services, the capability to share data with different stakeholders is becoming an important feature in ICT systems and enables new user participation models. The convergence of technologies such as cloud computing, Internet of Things (IoT) and big data will give rise to novel smart applications. It will transform how ICT services are integrated in our everyday business and private life, affecting all domains – including sensitive ones with high security and privacy needs.

Monetary and scalability benefits have driven the adoption of cloud storage solutions while their security still remains questionable. Recent leaks about espionage affairs and large-scale data breaches[1] increased the public perception of the importance of data privacy and security. Both issues are paramount for achieving users' acceptance. Approaches to strengthen the security of cloud-storage solutions – and thus increase user's trust – have already been proposed [1], but open issues remain.

We present a new framework for secure and agile data storage and sharing in the cloud without any single point of trust or failure. It is based on an distributed architecture and our system provides full multi-user support and enables advanced sharing scenarios without complex key management and revocation mechanisms.

[1]http://www.theguardian.com/us-news/the-nsa-files, accessed 2015-06-19.

Outline. In section II we present the problem of cloud based information sharing and discuss the requirements and desired security goals. In section III we discuss one possible approach which applies secret sharing to distributed systems in order to increase the security of stored data and review the state of the art. In section IV we present the ARCHISTAR framework and it's design goals and underlying architecture. Our open source implementation and first test results are presented in section V followed by an outlook and open issues in section VI.

II. Secure Data Sharing

The intrinsic multi-tenancy of cloud computing and the broad connectivity introduces new security threats, with tremendous risk for sensitive data (e.g., [2], [3]). The most appealing delivery model is also the most vulnerable: public clouds. The two major concerns as defined by Cloud Security Alliance[2] are 1) data breach and 2) data loss. Besides external threats, and if no additional cryptography is used by the customer to protect the confidentiality and integrity of data, the cloud provider needs to be fully trusted to rule out insider threats.

Requirements for cloud based data sharing are 1) that a data owner must be able to dynamically authorize user access to their data with fine granularity, 2) no unauthorized user (including the cloud provider) must be able to access the data, 3) authorized user access should be possible at any time without requiring the data owner's intervention, 4) data owners must be able to revoke access to data for any authorized user, and 5) no authorized users must be able to grant or revoke access rights (unless explicitly allowed to do so by the owner).

Building such a cloud based solution requires the combination of various techniques ranging from authentication mechanisms to access control systems up to strong isolation between different tenants. Strong protection for user data can only be achieved through encryption and ideally end-to-end protection is provided. Cloud storage providers who cannot access user data without customers' consent or authorization are often denoted *zero-knowledge*[3] and are considered the most trusted among all offerings.

[2]https://cloudsecurityalliance.org/download/the-notorious-nine-cloud-computing-top-threats-in-2013, accessed 2015-06-22

[3]We emphasize that this must not be confused with the zero-knowledge property of (interactive) proof or argument systems.

A. *Encrypted Cloud Storage*

Limiting oneself to established standard cryptographic methods complicates the integration of data-sharing and encryption. Application of existing primitives from symmetric and public key cryptography introduces complex key management. This leads to scalability issues as well as serious problems when it comes to access right revocation. In particular, if an action is executed, e.g., granting or revoking access rights, the user performing this actions must be online. In order to illustrate the problem with such a solution, we sketch the procedure of adding and revoking a member of a group:[4] to add a member, the user has to retrieve the random symmetric key and encrypt it again under the new user's public key. To revoke access for a particular user (or only his key in case of loss or exposure), the data owner must perform a re-encryption. In particular, he has to retrieve and remove all cloud-stored data, decrypt and subsequently encrypt it with a new random key before uploading the new data and keys to the cloud. The computation and communication overhead is significant as all computation must be performed client-side. This limits its usability for mobile or low-power clients. We note that although typically lazy revocation [4] is applied, i.e., re-encryption is delayed to the next write operation, the effort still can be far too high. Furthermore, this approach requires a public key infrastructure (PKI) to be available to all users.

Recent approaches, e.g,. [5], solve some of the problems associated with this use case. Most prominently discussed are currently *attribute-based encryption* [6], [7] and *proxy re-cryptography* [8], [9], which we briefly discuss below. In contrast, this work presents a new approach based on technologies originally intended to protect data in a cloud-of-clouds setting (for a single user-setting), but extended to a *multi-user distributed storage system.*

B. *Attribute-Based Encryption*

Attribute-based encryption (ABE) [6], [7] is a recent public key encryption paradigm that allows for end-to-end secure data storage and sharing systems [10], [11].

In ciphertext policy ABE (CP-ABE) approaches [12], a user's private key is associated with a set of attributes and a ciphertext specifies an access policy over a universe of attributes. A private key holder is able to decrypt a ciphertext if and only if his attributes satisfy the policy of a ciphertext. CP-ABE thus allows to realize implicit authorization, i.e., authorization is included into the encrypted data and only people who satisfy the associated policy can decrypt data. People who may decrypt may not be known when producing ciphertexts. In a data sharing scenario, keys can be issued by the data owner or some other authority. Recent works, e.g., [13], consider ABE in data-sharing scenarios and investigate schemes where ciphertexts can be re-encrypted to stricter policies and also efficient constructions for revocable ABE.

C. *Proxy Re-Cryptography*

The use of proxy re-cryptography [8], [9] can also overcome the limitations of content encryption in data-sharing applications. The basic idea behind proxy re-encryption is that one can give a re-encryption key to a semi-trusted proxy that allows the proxy to re-encrypt data encrypted under one key into data encrypted under another key without seeing the plaintext. Consequently, re-encryption can be outsourced, although meaningful (and efficient) proxy re-encryption schemes only exist for the public key setting. Thus the application of such schemes does not eliminate all problems within revocation as discussed above.

III. The Cloud-Of-Clouds Approach

An alternative approach to encryption of data in distributed systems is basing confidentiality on the use of secret sharing [14], which in addition to confidentiality also provides increased availability. Secret sharing algorithms split data into n pieces (shares) which are distributed to multiple storage providers. The shares needed to reconstruct the original data are an arbitrary monotone subset of the distributed original shares, e.g., a configurable threshold $k \leq n$ in the simplest case. Any malicious adversary, i.e., an external attacker or a storage provider, holding less shares than required for reconstruction, cannot reconstruct the original data. It is possible to resist fail-stop events or even active attacks without service interruption or losing data integrity by the combination of secret sharing with Byzantine agreement protocols.

A. *Advantages*

Through their inherently distributed nature, secret sharing based storage schemes provide means for increased data confidentiality, integrity and availability. This so called cloud-of-clouds approach is very appealing in practical implementations, as it prevents data breaches and data loss at the same time. If built without any single point of failure or trust it additionally mitigates vendor lock-in.

Today's service-level agreements (SLAs) for cloud storage capacity typically provide guarantees in terms of uptime, but specify little in terms of data availability or data protection. A typical value for service availability is 99.9%, e.g., as provided by Amazon web services (AWS), one of the leading public cloud platforms. It defines this under the term "Service commitment" and makes Amazon S3 available with a "Monthly Uptime Percentage" of at least 99.9% during any monthly billing cycle [5]. But *nothing* is said about guarantees for data loss or confidentiality. Public cloud offerings delegate the responsibility of protecting against data loss or data breaches to users. To be on the safe side they should encrypt data and replicate it to different locations, which

[4]We assume the use hybrid encryption, i.e., to use an efficient symmetric encryption scheme to encrypt the content data with a key chosen uniformly at random and then store the encrypted content data along with the random key encrypted under the public keys of the members of the group the data is shared with.

[5]https://aws.amazon.com/s3/sla/, accessed 2015-06-19.

Table I
FAILURE PROBABILITY FOR DIFFERENT CONFIGURATIONS

k/n	2/4	3/7	4/10
$\Pr(failure)$	$4.0 \cdot 10^{-9}$	$2.1 \cdot 10^{-14}$	$1.1 \cdot 10^{-16}$

means that users need to use different providers and thus ends up in a distributed setup.

A secret sharing based architecture has this property by design and due to the flexibility of the reconstruction threshold can be more efficient than simple replication. To demonstrate this behavior, we analyze the system in a classical reliability model where the overall system availability is calculated by the binomial cumulative distribution function (CDF).

$$\Pr(avail) := \Pr(X \leq (n-k)) = \sum_{i=0}^{\lfloor (n-k) \rfloor} \binom{n}{i} p^i (1-p)^{n-i}$$

The random variable X is the number of nodes which are not available during a single transaction and the failure probability of a single provider is $p = 0.001$ (99.9% SLA). For a system where n providers each get a single share of the data and with k being the number of shares required to reconstruct the data, the overall failure probability is given by the probability that more than $n-k$ nodes fail during a transaction, thus $\Pr(failure) := 1 - \Pr(avail)$. The figures to demonstrate the favorable behavior for typical configurations of our system in the case of fail-stop nodes are shown in Table I.

Another advantage of secret sharing based storage systems is that there is no need for explicit encryption keys and therefore key management issues do not emerge. The keyless property make them a very appealing candidate to build cloud based information sharing systems with increased dynamicity and capabilities for advanced sharing scenarios. Because secret sharing is information theoretical secure, the strong confidentiality properties can be used to build storage systems which are long-term secure [15].

B. Disadvantages

There are two major disadvantages of secure distributed cloud systems: while well understood and provable secure, the use of pure secret sharing introduces a different notion of security than known from encryption schemes. The security model requires an explicit non-collusion assumption and the guarantees are only true for a limited number of colluding nodes (providers). Without additional methods, it is not possible to protect or detect breaches of any kind if k or more providers collude. However, additional methods can be integrated to relax this property and add detection mechanisms for integrity breaches.

In addition, distributed systems are more complex to implement and add additional administrative overhead. In the case of cloud storage, this disadvantage is less dramatic as it relieves users from the complexity of designing hot-failover systems. Even non-distributed systems must include backup storage locations to protect against data loss. If availability is a concern, single points of failure must be prevented. Hot-backup sites with full state-replication for immediate fault-recovery lead to systems that are comparable in complexity and costs to secret sharing systems.

C. State of the Art

Various secret sharing based storage systems have been proposed and implemented in the last years (cf. [1]). We will briefly summarize the most relevant ones and discuss their shortcomings when it comes to their applicability as cloud based information sharing systems.

TahoeLAFS [16] encrypts data, disperses the encrypted data through erasure-coding and stores the fragments on multiple active backend storage servers. The storage servers themselves can utilize locally attached cloud storage for their storage backend. It employs a proxy-pattern where only a single proxy can access the stored data at a time.

Nubisave [17] allows for flexible combination of operations upon user data. Incoming user data can be deduplicated, erasure-coded and then distributed upon multiple cloud storage providers. Cloud providers can be dynamically selected due to run-time criteria. Operations are mostly implemented though external libraries, the `splitter-ng` component can perform limited secret sharing (Shamir) through the `JSharing` libary. Parallel access to backend data by multiple Nubisave instances is not supported.

RACS [18] utilizes erasure-coding to distribute data upon multiple cloud storage providers. Its use case is prevention of vendor-lock-in, security and privacy is of no concern, i.e., data is not encrypted). Access to clients as well as to backend storage is provided through an Amazon S3-compatible protocol. Parallel access to stored data through multiple RACS instances is achieved through usage of Apache Zookeeper, which implements the Zab primary-backup protocol for synchronization.

DEPSKY-CA [19] is a Byzantine fault-tolerant storage system. It provides availability and confidentiality through computational secret sharing [20]. In DEPSKY's system model, servers cannot communicate with each other, clients must synchronize access to files through a low-contention lock mechanism that uses cloud-backed lock files for synchronization. This mechanism is just obstruction-free and depends upon synchronized clocks. We would prefer a synchronization directive that is suitable for high-contention situations (multiple parallel writers) and is robust in face of live-locks and malicious clients.

Summarizing, existing proposals and solutions are dealing with information dispersal mechanisms for remote data storage. They are using secret sharing or similar techniques like all-or-nothing transforms to protect confidentiality. Some of them even deal with Byzantine robustness. To the best of our knowledge, however, none of them supports full concurrency for a multi-user environment, nor does any of them provide a solution to implement information sharing[6]. Additionally, none of the proposals discuss the interplay with access control mechanisms.

IV. The Architecture of ARCHISTAR

ARCHISTAR's architecture is intended to build distributed storage systems for secure and resilient cloud based data sharing. It focuses on data availability and integrity as well as confidentiality guarantees, even when considering the storage providers as adversaries.

ARCHISTAR integrates secret sharing with protocols for Byzantine robustness in a multi-cloud setup. To enable secure agile collaboration between stakeholders, it integrates an authentication and authorization layer. The combination of these three components leads to a new type of system not yet considered in this context. It provides a fully decentralized multi-user system without any single point of failure or trust and increases attacker's effort significantly compared to other systems.

ARCHISTAR's high-level physical structure is shown in Figure 1a. Active clients represent user devices which encode/decode user-data with a configurable secret sharing scheme. This prevents servers from ever touching plain-text data and therefore increases the level of security. The use of multiple sharing schemes allow for a configurable trade-off between security and required communication bandwidth for different client scenarios ranging from enterprise network access to mobile clients.

On the cloud provider side, the architecture foresees the use of two elements – *passive* storage and *active* server components with the latter running in virtual machines. Active servers are mandatory for the execution of the Byzantine fault-tolerant protocols and are necessary to validate user credentials and enforce access control to stored data shares. They do not rely on persistent data and should be able to execute from a read-only certified program image while volatile data is only persisted to the passive storage layer. Through this separation of active and passive components we achieve flexibility with regard to storage-provider requirements. We assume communication links between servers to be of lower latency when compared to the client-server links, which is a reasonable assumption in a practical setting.

The node architecture is shown in Figure 1b. Users access the node through the Client Application Programming Interface (Client API) which is highly influenced by the Amazon S3 access protocol. Node-to-node communication is performed through a Node API but both node and client control flow is unified within the node. All submitted commands are subject to the Authentication and Authorization component that verifies user's identity and their corresponding access rights. Required user and object information can be gathered from the User Policy Database (UPDB), object access information is stored within the Object Policy Database (OPDB). All operations, including accessing UPDB and OPDB, are synchronized through the BFT distribution component before they are finally executed within the Application Logic Component. To allow for modularity all data storage access is managed by minimal storage system interfaces ("Storage Int.").

The architecture does not depend on special modifications of cloud offerings, but rather is intended as an overlay or virtual cloud service on top of existing cloud offerings. This approach is preferable to approaches which require modifications in the used cloud stack.

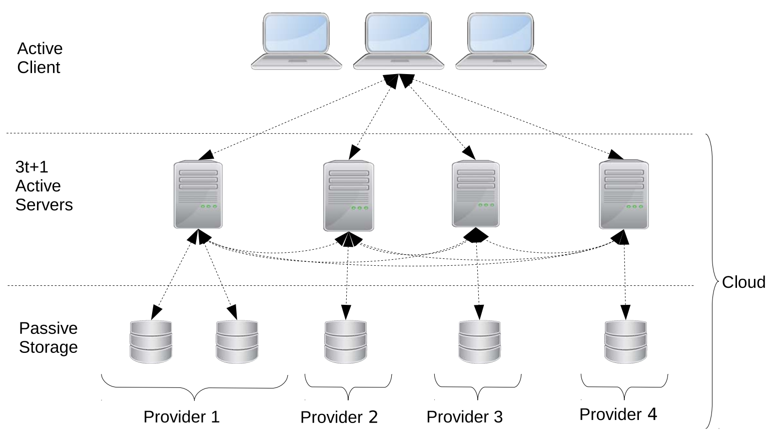
A. Byzantine Robustness and Concurrent Access

Within Byzantine fault-tolerant (BFT) systems a defined number of components may fail arbitrarily without impacting overall system stability. Initially thought to be too inefficient for general use, *practical fault tolerance* [22] (PBFT) sparkled a renaissance of BFT algorithms in 2001. This algorithm was well suited for asynchronous networks with limited (lower than a third) malicious participants and introduced multiple optional performance-enhancing additions. Subsequent BFT protocols reduce the minimal replica count and thus improve performance by either limiting the attacker's possibilities [23], [24] or introducing the concept of speculative BFT [25], [26]. The latter protocols assume the majority of operations to be non-faulty and introduce optimized protocols for this case. During faults, those protocols commonly fall back to PBFT. Recent security analysis [27], [28] indicates that while malicious errors do not prevent completion of those protocols, they might remove any upper-bound on operation latency.

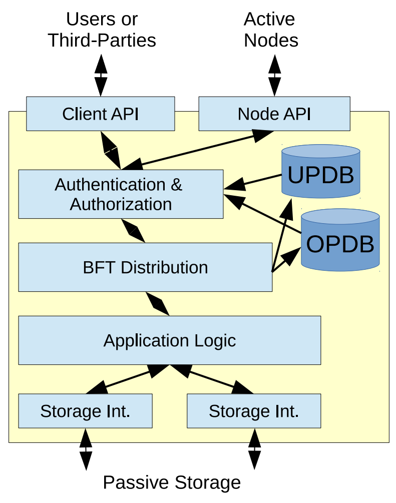
ARCHISTAR employs a Byzantine fault-tolerant distribution algorithm akin to PBFT. PBFT assumes that each server has full access to the stored plain-text data. As this assumption does not hold within a secret sharing storage system we had to adopt the protocol. Clients perform secret sharing locally and pass on an encrypted share to each node. In contrast, in PBFT the primary node receives the initial client request and forwards it to all nodes – this would concentrate all data at the primary and thus break the privacy gained through secret sharing. We cannot create an operation hash for operation matching, but depend upon an user-supplied hash value as identifier. In PBFT, the client performs a majority voting upon the operation's result (each node returns the same result as all data is identical between nodes). In a secret shared storage network, the data stored on nodes is not identical. Thus, we cannot perform meaningful comparisons upon operation's results. We opted for the initial simplistic low-performance option of the client performing a read operation after each data modification operation. Automated recovery/state transfer and their impact on privacy are still active research areas within ARCHISTAR.

The Byzantine fault-tolerant nature of the distribution scheme allows for malicious server components and partly Byzantine clients. In its current form, clients can submit inconsistent shares and their inconsistency will only be detected when those are reconstructed by another client. ARCHISTAR employs a simple versioning scheme to allow users to manually retrieve an older version of the

[6]We note, however, that there are solutions for the non-distributed case, e.g., [21].



(a) System



(b) Node

Figure 1. Architectural Overview

stored data. Future versions of ARCHISTAR will use the versioning information to allow for more granularity and improved parallel throughput.

As all access to the backend storage is synchronized through the BFT layer, ARCHISTAR can inherently deal with parallel concurrent write requests.

B. *Integration of Secret Sharing*

The ARCHISTAR architecture can be adapted to different needs and therefore supports multiple levels of confidentiality protection, namely information theoretical security, computational security or null confidentiality. The confidentiality levels can be combined with integrity checking mechanisms to increase robustness against active attacks. The supported modes of encoding are summarized in Table II.

Given a BFT system which is capable of coping with t malicious nodes, one requires at least $3t + 1$ nodes in the system. BFT algorithms are designed to replicate transactions and data in a consistent way throughout the system. The fact that they require $3t+1$ nodes is governed by liveness requirements and the assumption that the adversary has full control over the communication path. However, the last step of the protocol always involves a majority voting of at least $2t + 1$ answers from the system by the client. When secret sharing is integrated with BFT, various problems arise. Data consistency cannot be checked within the system anymore and majority voting cannot be applied to recover the data. Instead, reconstruction procedures have to be used. Furthermore, the threshold for the sharing cannot be selected freely and has to satisfy certain conditions, i.e., $k > t$ for confidentiality.

In order to cope, we separated transaction from data consistency. Transactions are always consistent, while the client-provided transaction payload, i.e., client data, is not consistency-checked during transaction execution but delayed until a client or auditor tries to reconstruct the data. A malicious client thus can corrupt single shares (until reconstruction), but not the overall system state. ARCHISTAR employs versioning to deal with this delayed inconsistency detection through a roll-back to a known consistent share version. This combination enables us to cope with arbitrary malicious clients in a very efficient way and without the use of verifiable secret sharing (VSS) schemes [29].

Standard secret sharing primitives can correct up to $\lfloor \frac{n-k}{2} \rfloor$ errors, for an optimal value of $k = t + 1$ the reconstruction procedure requires at least $3t + 1$ shares to successfully complete. Thus, t additional answers are required in the last phase of the BFT protocol and the amount of overall system nodes increases to $n \geq 4t + 1$. The only possibility to reduce the amount of system nodes is by the application of robust secret sharing (RSS), e.g., [30], which can reconstruct from $2t + 1$ shares correctly but introduces both, computational and storage overhead. For configurations with mandatory low node numbers, we are also supporting RSS which lead to the same $3t + 1$ requirements as plain BFT in terms of number of nodes.

As mentioned, ARCHISTAR provides multiple secret sharing algorithms. Shamir secret sharing [14] was selected as instantiation of perfectly secure secret sharing (PSS), i.e., to achieve information theoretical security. It can be used to provide a long-term security, but it is also used as a sub-protocol in more efficient schemes. For storage efficiency reasons, ARCHISTAR also provides computational secret sharing (CSS) based on methods presented by Krawczyk [20]. These methods encrypt the content with a symmetric encryption scheme and generate shares by the application of an information dispersal algorithm (IDS) [31], which is very similar to a Reed-Solomon code. The encryption key is shared along with the data by means of PSS[7].

[7]The current architecture defines AES-CBC and Salsa20 for encryption. AES was selected to have a trusted algorithm and Salsa20 as a trustworthy solution optimized for performance on embedded ARM hardware.

Table II
SUPPORTED MODES OF ENCRYPTION

Basic Modes	Description	Example Suite
M1	Null confidentiality (BFT)	None
M2	Perfectly secure secret sharing (PSS)	ShamirPSS
M3	Robust PSS (RPSS)	ShamirPSS-Poly1305
M4	Computational secure secret sharing (CSS)	ShamirPSS-ChaCha20
M5	Robust CSS (RCSS)	ShamirPSS-ChaCha20-Poly1305

To generate robust version of both PSS and CSS we employ information checking (IC) as proposed in [29] for settings with small number of nodes and optionally provide the scheme presented in [30]. Currently, we do not consider methods as presented in [32], which correct up to $2t + 2$ and also do not apply distributed fingerprinting as proposed by Krawczyk [20] in order to reduce the number of implemented algorithms.

C. Authentication and Authorization

Secure distributed storage systems have been proposed at various occasions, but have not been considered for data sharing and, to the best of our knowledge, no system provides all features necessary to allow for full multi-user concurrency. Moreover, it is necessary to investigate the interplay with authentication and authorization mechanisms (AA), since this is important for data confidentiality in a system where security is achieved by a non-collusion assumption.

To protect the information from unauthorized external access, strong authentication methods must be put in place at each node. There are no specific requirements for ARCHISTAR, except that the credentials must be different at each provider to avoid them to impersonate the legitimate user. To protect data from unauthorized internal access through legitimate clients, we propose a special authorization mechanism which is tightly coupled to the BFT functionality and provides a novel integrated approach enabling a fully decentralized data collaboration system without single point of trust or failure.

As illustrated in Figure 1b, every node integrates a dedicated stand-alone authentication and authorization module on top of the BFT layer. It has only read access to the user policy database (UPDB) containing the credentials and the object policy database (OPDB) maintaining the access control information for data objects. In our basic model, the user database is maintained at each node and all changes to the user database are made on all nodes through the Byzantine layer in plaintext mode. Using the BFT layer allows for globally consistent user and access control information which cannot be influenced by malicious nodes or clients. As this policy is enforced by the local authentication and authorization modules at honest nodes, malicious nodes or clients will never get enough shares to compromise the system. Even if the adversary controls all malicious nodes and clients in a coordinated way, she will not be able to retrieve enough shares.

Currently, we define a basic access control model where administrators can create, read, update and delete user and group accounts in the UPDB and object owners can attribute access rights with their objects within the OPDB. All information is stored in plaintext along the secret shared data using the Byzantine layer. This new architecture enables a new class of information sharing networks and although only simple access patterns have been designed so far, due to the nature of the systems with active nodes, which could run arbitrary code, many new access patterns will be possible. When using conventional encryption schemes, access control is implicitly enforced by the scheme and authorized users must be in possession of the right key to read the information. The key-less nature of secret sharing, however, enables a lot more flexibility and agility in data sharing, but it must be combined with dedicated AA system.

V. OPEN SOURCE FRAMEWORK

The open-source ARCHISTAR[8] framework implements large parts of the discussed architecture. Effort was spent to create generic components, which can be used in different use cases and future projects. We briefly summarize the major components and report on measured performance figures.

A. Archistar-CORE

The `archistar-core` project is the high-level project that incorporates all other libraries into the distributed storage prototype. Clients can access the storage system through an S3-compatible HTTP interface. Backend storage can be in-memory, filesystem-based or attached through S3-compatible cloud providers. Being the top-level project, automated integration tests are performed with a special testing instance which bootstraps four active BFT nodes with temporary in-memory databases on a single host.

The following libraries have been utilized:

- `archistar-bft`: the distributed BFT engine
- `archistar-smc`: the secret sharing algorithms
- `bouncy-castle`: traditional cryptographic primitives
- `netty.io`: network and thread management
- `RestEasy`: helper for Amazon S3-compatible client
- `JetS3t`: utilized for S3-based storage backends

Table III shows some line-of-code numbers for the projects. Refactoring was done to keep the amount of code

[8]http://github.com/archistar

Table III
LINES-OF-CODE FOR DIFFERENT COMPONENTS

Component	Code	Comments	Test-Cases
`archistar-core`	2254	674	586
`archistar-bft`	988	267	172
`archistar-smc`	2706	1528	1979

Table IV
SECRET SHARING PERFORMANCE

Algorithm	Share	Combine
ShamirPSS	31629.3	41541.6
RabinIDS	80629.9	92044.9
KrawczykCSS		
with AES-CBC	48075.1	48416.1
with AES-GCM	32741.8	33085.6
with ChaCha20	56109.6	67286.1

to a meaningful minimum. It should compare favorable to other BFT projects, which are said to be in the tens of thousands of code [33].

B. Archistar-SMC

The `archistar-smc` library implements multiple secret sharing algorithms. It abstracts implementation details behind two important interfaces (*SecretSharing* and *InformationChecking*) and thus allows for easy reconfiguration of implemented algorithms. The BouncyCastle library[9] is used as algorithm-provider because the Standard Java CryptoAPI does not provide sufficient error information during decryption.

Table IV shows performance numbers obtained from sharing and recombining data with a 4/3 split on the single core of an Intel Mobile Core i7-4600U (@2.1Ghz) under Java 1.8. All numbers are given as kByte/sec of "raw" (unencrypted) data. More precisely, when splitting data the resulting network bandwidth depends on the algorithm, e.g., four times when using ShamirPSS.

Looking at the results, "combine" performance is higher than "split" performance. As the combine-operation is used during read operations and those are suspected to be the majority of operations, this is advantageous. KrawczykCSS performance is mostly symmetric: this is to be expected as it encrypts the raw data with symmetric encryption schemes and then distributes the encryption key using ShamirPSS. The encrypted data is distributed through RabinIDS. Performance improvements within the encryption algorithm or within RabinIDS will automatically improve KrawczykCSS performance. We have tested the performance of AES-GCM vs ChaCha20. The former is said to be slow if no hardware acceleration is available, while the latter was optimized for use within software. As Java does not utilize the CPU's hardware acceleration, this can be seen in difference of a factor of two. This will be important for mobile devices as those are currently lacking hardware acceleration for AES.

[9]https://www.bouncycastle.org/java.html

Table V
INFORMATION CHECKING OVERHEAD

Rabin-Ben-Or	Blocks/sec	kByte/sec
with SHA256 HMAC	12.5	6480.0
with Poly1305	38.8	18919.0

Table VI
PERFORMANCE OF SHAMIRPSS-CHACHA20-POLY1305

n/k	Split	Combine
4/3	17210.1	14840.6
7/3	7148.3	7314.3

Information checking (IC) enhances secret sharing with distributed integrity protection. The algorithm are inherently symmetric due to the use of message authentication codes (MACs). Performance depends on the performance of the used MAC-Algorithm, which is fed using a BouncyCastle RNG. Given 512kByte data blocks, Table V shows how many unencrypted blocks/second and corresponding unencrypted raw data rate in kbytes/sec can be achieved. IC is the limiting step for sharing/combining. An alternative to IC would be the use of an error decoder while increasing the share count.

Finally, Table VI shows the achieved performance combining a KrawzywkCSS using ChaCha20 with information checking à la Rabin-Ben-Or using Poly1305.

We examined Cevallos et al.'s scheme [30] as alternative to the Rabin-Ben-Or scheme. It trades space (smaller check sums) with increased computational needs (error instead of erasure decoding). In our use cases the space reduction is too insignificant, especially when using data blocks in the MB range to justify usage of their scheme.

C. Archistar-BFT

We opted for a slightly modified PBFT algorithm with alterations to better suite our secret sharing use case. Alternatives like Zyzzyva [30] are thought to achieve better performance during "normal" operation, but degrade substantially during attacks [28]. We did not consider algorithms with special hardware or hosting requirements such as CheapBFT [23].

The `archistar-bft` component encapsulates a single BFT state machine as implemented by each active server/replica. All interactions with other servers are implemented through callbacks (inversion-of-control pattern), which allow for improved testability.

VI. CONCLUSION

In this work we present a new type of secure cloud based data sharing by combining Byzantine fault-tolerant techniques with secret sharing and adding authentication and authorization. It enables the deployment of a secure storage system for multiple users on top of existing cloud infrastructure in a distributed setup without single point of trust or failure. No colluding subset of providers below a defined threshold will be able to compromise data confidentiality or integrity. Nor will they be able to influence the system's availability or provoke data loss.

We present the corresponding Open Source framework, which serves as a reference system and can be used to implement prototypes. The system provides the basic functionality up to the BFT layer and further extension with authorization and authentication is ongoing work.

Future work will focus on two directions: on one hand, we will increase system dynamicity and add monitoring capabilities to efficiently manage configurations and system trust as well as assurance levels. On the other hand, research will be conducted to increase the privacy properties of the system and reduce the amount of metadata available to the providers.

Acknowledgment

This work has been supported by the Austrian Research Promotion Agency (FFG) through project ARCHISTAR under grant agreement number 832145, and European Union's Horizon 2020 Programme through project PRISMACLOUD under grant agreement number 644962.

References

[1] D. Slamanig and C. Hanser, "On cloud storage and the cloud of clouds approach," in *ICITST 2012*, 2012, pp. 649–655.

[2] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds," in *ACM CCS 2009*, 2009, pp. 199–212.

[3] Y. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Cross-tenant side-channel attacks in paas clouds," in *ACM CCS 2014*, 2014, pp. 990–1003.

[4] M. Kallahalla, E. Riedel, R. Swaminathan, Q. Wang, and K. Fu, "Plutus: Scalable secure file sharing on untrusted storage," in *FAST '03*, 2003.

[5] S. Yu, C. Wang, K. Ren, and W. Lou, "Achieving secure, scalable, and fine-grained data access control in cloud computing," in *INFOCOM 2010*, 2010, pp. 534–542.

[6] A. Sahai and B. Waters, "Fuzzy identity-based encryption," in *Advances in Cryptology - EUROCRYPT 2005*, 2005, pp. 457–473.

[7] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *ACM CCS 2006*, 2006, pp. 89–98.

[8] M. Blaze, G. Bleumer, and M. Strauss, "Divertible protocols and atomic proxy cryptography," in *Advances in Cryptology - EUROCRYPT '98*, 1998, pp. 127–144.

[9] G. Ateniese, K. Fu, M. Green, and S. Hohenberger, "Improved Proxy Re-Encryption Schemes with Applications to Secure Distributed Storage," in *NDSS 2005*, 2005.

[10] M. Pirretti, P. Traynor, P. McDaniel, and B. Waters, "Secure attribute-based systems," *Journal of Computer Security*, vol. 18, no. 5, pp. 799–837, 2010.

[11] S. Kamara and K. Lauter, "Cryptographic Cloud Storage," in *Financial Cryptography Workshops*, 2010, pp. 136–149.

[12] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *IEEE S&P 2007*, 2007, pp. 321–334.

[13] A. Sahai, H. Seyalioglu, and B. Waters, "Dynamic Credentials and Ciphertext Delegation for Attribute-Based Encryption," in *Advances in Cryptology - CRYPTO 2012*, 2012, pp. 199–217.

[14] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[15] J. Braun, J. Buchmann, C. Mullan, and A. Wiesmaier, "Long term confidentiality: a survey," *Designs, Codes and Cryptography*, vol. 71, no. 3, pp. 459–478, 2014.

[16] M. Selimi and F. Freitag, "Tahoe-lafs distributed storage service in community network clouds," in *BdCloud 2014*. IEEE, 2014, pp. 17–24.

[17] J. Spillner, G. Bombach, S. Matthischke, J. Muller, R. Tzschichholz, and A. Schill, "Information dispersion over redundant arrays of optimal cloud storage for desktop users," in *UCC 2011*. IEEE, 2011, pp. 1–8.

[18] H. Abu-Libdeh, L. Princehouse, and H. Weatherspoon, "Racs: a case for cloud storage diversity," in *1st ACM Symposium on Cloud Computing*. ACM, 2010, pp. 229–240.

[19] A. Bessani, M. Correia, B. Quaresma, F. André, and P. Sousa, "Depsky: dependable and secure storage in a cloud-of-clouds," *ACM Transactions on Storage (TOS)*, vol. 9, no. 4, p. 12, 2013.

[20] H. Krawczyk, "Secret sharing made short," in *Advances in Cryptology - CRYPTO '93*, 1993, pp. 136–146.

[21] R. A. Popa, J. R. Lorch, D. Molnar, H. J. Wang, and L. Zhuang, "Enabling Security in Cloud Storage SLAs with CloudProof," in *2011 USENIX Annual Technical Conference*, 2011.

[22] M. Castro and B. Liskov, "Practical byzantine fault tolerance and proactive recovery," *ACM Trans. Comput. Syst.*, vol. 20, no. 4, pp. 398–461, Nov. 2002.

[23] D. Zhuo, Q. Zhang, D. R. Ports, A. Krishnamurthy, and T. Anderson, "Machine fault tolerance for reliable datacenter systems," in *APSys '14*. ACM, 2014, p. 3.

[24] M. Correia, N. F. Neves, and P. Verissimo, "Bft-to: Intrusion tolerance with less replicas," *The Computer Journal*, 2012.

[25] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong, "Zyzzyva: Speculative byzantine fault tolerance," in *SOSP '07*. ACM, 2007, pp. 45–58.

[26] G. Santos Veronese, M. Correia, A. N. Bessani, and L. C. Lung, "Spin one's wheels? byzantine fault tolerance with a spinning primary," in *SRDS'09*. IEEE, 2009, pp. 135–144.

[27] A. Clement, M. Marchetti, E. Wong, L. Alvisi, and M. Dahlin, "Making byzantine fault tolerant systems tolerate byzantine faults," in *NSDI'09*, 2009.

[28] A. Singh, T. Das, P. Maniatis, P. Druschel, and T. Roscoe, "Bft protocols under fire." in *NSDI*, vol. 8, 2008, pp. 189–204.

[29] T. Rabin and M. Ben-Or, "Verifiable secret sharing and multiparty protocols with honest majority," in *STOC '89*. ACM Press, 1989, pp. 73–85.

[30] A. Cevallos, S. Fehr, R. Ostrovsky, and Y. Rabani, "Unconditionally-Secure Robust Secret Sharing with Compact Shares," *Advances in Cryptology - EUROCRYPT 2012*, pp. 195–208, 2012.

[31] M. O. Rabin, "Efficient dispersal of information for security, load balancing, and fault tolerance," *Journal of the ACM*, vol. 36, no. 2, pp. 335–348, 1989.

[32] M. Jhanwar and R. Safavi-Naini, "Unconditionally-Secure Robust Secret Sharing with Minimum Share Size," in *Financial Cryptography*, 2013, pp. 96–110.

[33] R. Guerraoui, N. Knežević, V. Quéma, and M. Vukolić, "The next 700 bft protocols," in *EuroSys '10*. ACM, 2010, pp. 363–376.