Secure Access Control for Multi-Cloud Resources

Hendrik Graupner, Kennedy Torkura, Philipp Berger and Christoph Meinel
Chair of Internet Technologies and Systems
Hasso Plattner Institute, University of Potsdam
Potsdam, Germany
Email: {hendrik.graupner, kennedy.torkura, philipp.berger, christoph.meinel}@hpi.de

Maxim Schnjakin
Innovations Department
Bundesdruckerei GmbH
Berlin, Germany
Email: maxim.schnjakin@bdr.de

Abstract—**Privacy, security, and trust concerns are continuously hindering the growth of cloud computing despite its attractive features. To mitigate these concerns, an emerging approach targets the use of multi-cloud architectures to achieve portability and reduce cost. Multi-cloud architectures however suffer several challenges including inadequate cross-provider APIs, insufficient support from cloud service providers, and especially non-unified access control mechanisms. Consequently, the available multi-cloud proposals are unhandy or insecure. This paper proposes two contributions. At first, we survey existing cloud storage provider interfaces. Following, we propose a novel technique that deals with the challenges of connecting modern authentication standards and multiple cloud authorization methods.**

Index Terms—**Cloud storage, access control management, multi-cloud systems, data security**

I. Introduction

Cloud storage is an evolving shift in computing especially as it provides storage services as a utility, via a pay-as-you-go model. The potentials of this paradigm are attractive, offering features such as cost saving, high scalability and flexibility [1]. Gartner[1] asserts that the bulk of information technology spending will be on cloud computing technologies (including cloud storage) by 2016. However, individuals and organizations have doubts on the security of cloud storage. Reasons for these doubts include reports of security breaches among even popular Cloud Service Providers (CSPs) [2]. More worrisome are the recent revelations of back-doors in cloud software and illegal handover of encryption keys to government agencies [3]. CSPs do not completely guarantee availability, confidentiality and integrity of customers' data. Accordingly, enterprises prefer alternative approaches like private clouds [4] rather than public clouds, even when this results to substantial increase in financial and administrative costs.

While several approaches have been proposed to improve the mentioned security aspects, a lasting solution is an open challenge. A proper approach consists in the use of multi-cloud storage platforms [5] [6]. This approach leverages on concurrently accessing multiple cloud storage to overcome data lock-in and availability issues. But, there are still bottlenecks hindering this approach from becoming an enduring solution. Differences in the architectures and implementation styles of the CSPs complicates the development of cross-provider Application Programming Interfaces (APIs). Hence, developers are left to develop applications without guarantees of support from the providers. Moreover, there are rarely established standards for cloud systems, this further complicates the issue. Furthermore, current multi-cloud APIs [7] do not focus on cloud security requirements such as access control procedures, authentication and authorization; which are major requirements for secure systems [8]. Cloud access control brokers are being favored to mitigate the mentioned issues especially for enterprises. Accordingly, we propose in this paper, a multi-cloud access control broker scheme suitable for an enterprise use-case. *Our solution allows precise access control over distributed data resources by providing an additional Access Control Management (ACM) service (see Figure 1).*

Our ACM is built on the CloudRAID system [9], an existing software system that conveys Redundant Array of Independent Discs (RAID) principle to the cloud as against its traditional employment in conventional data centers. CloudRAID leverages on erasure coding techniques to slice data into bits. These data bits are thereafter distributed across several storage repositories and similarly retrieved and reconstructed when required, albeit in a manner that achieves redundancy. This approach ensures availability and data protection in cloud storage, while ensuring optimal performance. This is achieved by parallel access to slices of the data spread across several cloud repositories. Hence, CloudRAID does not rely on the availability of individual CSPs and addresses the three major security obstacles in cloud computing [10], as follows: availability, data lock-in and confidentiality.

The utilization of multiple cloud storage repositories is a core principle of the CloudRAID concept. A proper management of access and resources is imperative for the security and integrity of users' data. To overcome the lack of standardized,

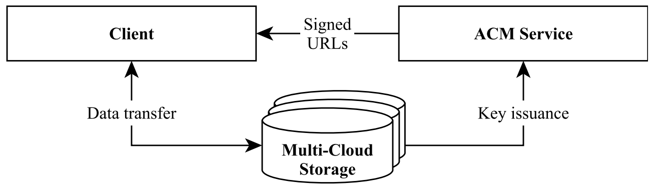


Fig. 1: Our *ACM service* compensates for the lack of standards in cloud storage access control by using Signed URLs for granting temporary access.

[1]https://www.gartner.com/newsroom/id/2613015, visited: 15-05-2015

unrestrained ACMs we propose a third party service leveraging the principle of Signed URLs. We believe that it could pave the way for an uniform, fine-grained access control strategy across different cloud storage repositories.

We aim to answer two important questions which are currently present in cloud computing research: *How can individual CSP authorization methods and common authentication standards be integrated by an access broker in a secure, performant, and extensible way? Which technologies ensure data confidentiality toward a semi-trusted access broker?*

In this paper we survey the current state of cloud computing ACMs and propose our approach on top of their APIs. Hence, the rest of this paper is structured as follows. We discuss preliminaries and works similar to our contribution in Section II. A technical survey of important features of popular CSPs such as ACMs and interoperability with other providers is conducted in Section III. Section IV proposes our access broker concept that enables single sign-on, cross-provider ACM and describes its integration into the CloudRAID system. An evaluation of some experiments performed in our implementation is highlighted in Section V, with further discussion on benefits and shortcomings of our approach in Section VI. We present ideas for future work based on our experience in Section VII.

II. Related Work

In this work, we aim at providing reliable access management of storage resources in a multi-cloud storage infrastructure. Multi-cloud infrastructures have been recommended as a possible solution to several concerns besieging cloud computing [5]. They afford cloud users flexible deployment of cloud resources and overcome vendor lock-in challenges [8].

MetaCDN [11] is a software designed to leverage cloud storage as a cheaper alternative to Content Delivery Networks (CDNs). Data is basically replicated across multiple cloud storage locations worldwide to achieve availability; erasure codes are not utilized in this work hence it becomes a rather expensive option. Furthermore, the security concerns of the cloud are not the focus of the authors. Hussam et al. introduce Redundant Array of Cloud Storage (RACS) [12], a proxy that employs a strategy similar to our system. RACS focuses on vendor lock-in and neglects the other security issues that impede cloud computing.

In [13] the authors introduce NCCloud, a storage system that employs network minimum-storage regenerating codes for achieving fault tolerance in a multi-cloud setting additional to storage repair. It is important to note, that the sustainability of methods employed are tested on a single cloud with the assumption that similar results are achievable for other CSPs.

Another closely related work to ours is DepSky-A [14]. The authors utilize quorum techniques to replicate data across several cloud repositories in order to enhance availability and integrity. This approach does not have much economic value since the cost for cloud storage is not reduced, it is almost twice the cost of using the traditional approach. Furthermore, the intelligent placement of data across the storage repositories using the most common parameters desired by clients is not a feature of DepSky-A, this is available in our system. On the overall, the above mentioned authors do not tackle the question of access management in multi-cloud systems as we do, in conformity with access control best practices in cloud computing. [8][15].

The research into multi-cloud infrastructure is growing, there is currently an increasing number of proposals being made especially to address the issue of harmonized standards and interoperability. CSPs have own-defined ways for accessing their storage resources, hence unifying access across several CSPs is challenging. Some CSPs now provide Software Development Kits (SDKs) and APIs for developers to configure their specific requirements, however this does not completely solve the issue of multi-cloud scenarios since the provided frameworks are not interoperable with other CSPs. While there are a couple of efforts on the question of unified APIs, there are still open challenges including the issue of language agnostic limitations [16] and secure access control methods.

Reimer et al. propose a pattern called ”Federated Identity Access Broker” [17] which involves a service handling authentication and authorization based on SAML 2.0 and OAuth 2.0. The question of user secure access to these cloud storage repositories also remains unanswered.

III. Survey of Cloud Storage Providers

In this section, we conduct a survey of selected cloud storage providers (summarized at Table I) in order to highlight the state-of-the-art in cloud access control techniques. The first set of providers are completely independent while the next set follow a common cloud strategy since they base most of their services on OpenStack cloud operating system. Leveraging on this information, we expose the difficulties in adopting a unified ACM for multi-cloud storage and discuss on possible options. Security analysis of the discussed access control techniques is beyond this paper’s scope.

A. Cloud Storage Providers

Google Cloud [18] has three storage offerings namely Cloud SQL, Cloud Datastore and Google Cloud Storage (GCS).

TABLE I: CSPs offer various ACM methods

Cloud Service Provider	Access Control Mechanisms
AWS S3	AWS IAM, Resource-based ACLs, User based ACLs, LDAP, pre-Signed URLs
Microsoft Azure	Azure Active Directory, LDAP, Shared Access Signatures (SASs), ACLs
Google Cloud Storage	ACLs , Signed URLs
OpenStack Swift	Temporary URLs, RBAC, LDAP
IBM SoftLayer	ACLs,Temporary URLs, LDAP
HP CloudFiles	General ACLs, Cross-Project ACLs, Temporary URLs
RackSpace CloudFiles	RBAC, ACLs, Temporary URLs

We focus on GCS, there are two major options available for accessing objects and containers; Access Control Lists (ACLs) and Signed URLs (Query String Authentication). Both approaches are not mutually-exclusive, both can be used at the same time. ACLs are used to provide access to container and objects, and also for sharing objects with other users. Containers have the default project-private ACL creation, hence owners have the responsibility of applying pre-defined ACLs to ensure security [19]. Similarly, objects created inherit the ACLs of their parent container unless a pre-defined ACL is applied. Roles and permissions (*read*, *write*, *full control* and *default*) can be specified for containers and objects, and applied to scopes (specific users or groups). Permissions are only possible for users having Google accounts and only 100 ACL entries can be specified per object and container. This feature places a limitation for multi-cloud access use-cases; where users don't necessarily wish to have Google accounts before accessing resources. Signed URLs access control method provide a "remedy" to this limitation, it facilitates access to be granted to GCS resources for non-Google accounts. Google provides APIs and various client libraries for accessing its cloud storage.

Microsoft Azure Storage Service consists of BLOB Storage, Table Storage, Queue Storage and File Storage. The storage service is managed through Microsoft accounts, a maximum of 20 sub-accounts is permissible per account. There is no limit to the number of containers and BLOB acquirable by a storage account other than the total storage being less than 1,000 terabytes. Azure containers are private by default when created, hence appropriate storage keys are necessary for access. Containers can be set to *public* for open access to everybody though access keys will be required to edit or delete blobs. Azure storage resources are also exposable through SAS, which is similar to the Signed URLs option of GCS. SAS enables clients that cannot be trusted with storage account credentials to be securely delegated granular access to storage resources [20]. Azure Active Directory is also used to control authentication and access on the Azure Storage platform.

Amazon Web Services (AWS) Cloud Storage flavors include Amazon Glacier, Amazon CloudFront, Amazon Elastic Computing Cloud (EC2) Block Storage Volumes, Amazon Elastic File System and Amazon Simple Storage Service (S3), we focus on S3. Interaction with S3 is possible through the AWS Management Console, Command Line Interface (CLI), SDKs and service-specific APIs. S3 resources are private by default, resource owners have full control and can grant access and permissions using access policies. Access policy mechanisms are broadly categorized into resource-based policies and user policies [21]. ACLs are automatically generated for all containers and objects on creation. Roles and permissions may also be assigned using cross-account access, identity federation, and Web Identity Federation. These methods are only possible for existing AWS accounts or through organizational identity and authentication systems (e.g Lightweight Directory Access Protocol (LDAP)) [22]. AWS ACLs do not suffice multi-cloud access control since grantees must be existing AWS accounts. AWS provides a method for granting temporary access through use of pre-Signed URLs. This technique is preferred for multi-cloud access control as described at Section IV.

OpenStack-based Cloud Storage Providers OpenStack is a open source cloud operating system that suits private, public and hybrid clouds [23]. It is particularly fashioned to be compliant with AWS S3, though it also comprises similar services with other providers. OpenStack's object storage is known as Swift, it can be managed through the OpenStack dashboard (Horizon) and CLI, programmatic management is possible using various SDKs. OpenStack KeyStone [24] handles authentication, authorization and access management, it also supports LDAP integration. Account owners can create an unlimited number of containers, which may be assigned to various child accounts or shared with different permission levels. This is much more feasible for multi-cloud implementations. Time limited access is also possible through the Temporary URL feature, which is similar to the GCS Signed URLs approach. OpenStack is a growing ecosystem, it is currently the reference implementation for some providers such as Hewlett-Packard (HP), RackSpace and IBM, hence the features available in OpenStack are available in these providers, though with a few differences. Adopting the access management provisions of OpenStack is a possible path for our cloud broker approach, however, we could be limited to the cloud providers that support OpenStack, thereby defeating our aim.

The storage providers discussed above adopt proprietary access control techniques, which are incompatible. Hence integration into a single, multi-cloud platform is challenging. However, since this is our goal, we consider possible options in the next sub-section.

B. Multi-Cloud Unified ACM Approaches

There are three options for cloud storage brokering:

- **One account per user per CSP.** In this approach, clients register for their desired service provider while the broker manages access and other aspects. However, the approach suffers from the diversity of storage providers' user account management and API design as earlier discussed. Programmatic user account creation is also not possible, and the broker is required to provide support for so many storage providers, which could introduce security and performance issues. In addition, users have to share their credentials with various providers, thereby exposing themselves to identity theft. From a storage brokerage perspective, this option amounts to loss of control over the storage due to end users being owners of the accounts, and limits some functionalities like file sharing.
- **CSP sub-account structures.** In this approach the sub-account structures provided by the CSPs are leveraged. However, these structures are limited in the number of users, e.g. AWS provides a sub-account structure for a fixed number of users and recommends some temporary access methods if more users are required. Therefore, a system that relies on provider-side ACM could only support providers that fully implement this concept, moreover,

such users are required to be existing users. This is not a viable option for a service with a huge client-base.

- **Single service account per CSP.** This option users seamless access to broker-owned containers hosted by storage providers. This is the option adopted by DropBox; which uses AWS S3 for storage at the backend. This option is the best since it overcomes the limitations of the above mentioned options. ACM can be implemented from scratch using the provider's APIs, e.g. OAuth2.0 [25] or Signed URLs. n the other hand, end users' data is highly under control of the account holder, which is the service operator. This is a moderate concern in a software architecture where the service provider can not read any contents, despite the ability of full access to any data stored.

IV. Implementation

This section describes a reusable approach of the successful integration of ACM interfaces across CSPs. The context is a service-oriented client-server-architecture with special focus on data confidentiality, availability, and service transparency.

A. Requirements

The requirements emerge from operations which client applications need to carry out. In cloud storage systems these are typically file upload, download, update, and deletion. Today's de facto standard for basic file operations on cloud storage is a REST-ful API based on the HTTP protocol. Hence, cloud based ACM must provide an according permission transfer.

It has to be considered that client applications may be replaced by malicious software. In order to prevent abuse of the permissions granted on the cloud storage, a high level of security has to be part of the technique of choice. Firstly, the duration of validity of permissions should be as short as technically possible. Client applications should never possess a token that is valid for an unlimited time. This guarantees the actual account owner a maximum of control over active permission tokens and the ability to perform authorization checks on a request base. Secondly, the permission should only be valid for a particular resource. I.e., on a cloud container the exact location has to be specified. For file upload this also means subsequently binding the access token to a local file. Possible parameters to ensure upload content integrity are content checksum (e.g. MD5), content length, and Internet media type. This measure reduces the risk of a potentially malicious client application to upload malware or affect data integrity.

B. Permission Transfer in Clouds

On the cloud end of an access broker, APIs for permission transfer are required. In Section III we discovered the following technologies to pass permissions on cloud resources to other parties:

- **Public access** is always possible but contradicts with the principle of data confidentiality. Hence, it is not an option in most scenarios.
- The most common technology among CSPs are **ACLs**. It describes a handy, yet not standardized way of defining permissions on cloud resources. The most significant shortcoming is that it requires permission recipients to have accounts with the CSP. This limitation and others, like Google's limitation of 100 entries[2] make it unfeasible in systems that aim support a big amount of users.
- The most flexible approach to share permissions is a technique of pre-signing an authorized operation on a resource. Since access to cloud storage typically happens on the HTTP protocol, HTTPS URLs are representing operation requests. Under different names, all surveyed CSP implement the technique of URL pre-signing. We use the term **Signed URLs** which is employed by Google. To implement a flexible, yet powerful ACM, we believe that Signed URLs are the most suitable technology.
- In addition to the technologies discussed above, most vendors implement their own, **provider specific** ways of ACM. Examples are Signed Policy Documents (GCS), Azure Active Directory (Microsoft), KeyStone (OpenStack), etc.

C. Signed URLs

Signed URLs, a form of query string authentication, is an access control feature used by most cloud storage services though with differences in name and implementation. It is referred to as pre-Signed URL in AWS, Signed URL in GCS, temporary URL in OpenStack and SAS in Microsoft Azure. *URL signing represents a concept of providing temporary access to specific resources.* The owner of a cloud account *signs* a certain HTTP URL by attaching a security token, that only the holder of a secret can create. The URL is then passed to a client application which uses it to make an authorized request to the storage provider.

The concept of Signed URLs is designed to fit into Service-Oriented Architectures (SOAs) (see Figure 2). All state is part of the HTTP request in form of query parameters and headers, which makes it RESTful. Hence, an ACM service built on this technique can operate on request base and does not necessarily need to communicate with the CSP.

The actual signing operation is based on a combination of a cryptographic hash function and encryption. Feasible are combinations with symmetric (e.g. HMAC-SHA-256) as well as asymmetric (e.g. RSA-SHA-256) cryptosystems. As a requirement the signing service has to hold a secret i.e. a symmetric or private key issued by the CSP. The following steps represent creation and validation of an asymmetrically Signed URL. While creation is a task of the third party service's ACM the cloud storage provider's ACM validates requests to resources.

[2]https://cloud.google.com/storage/docs/access-control#About-Access-Control-Lists, visited: 20-05-2015

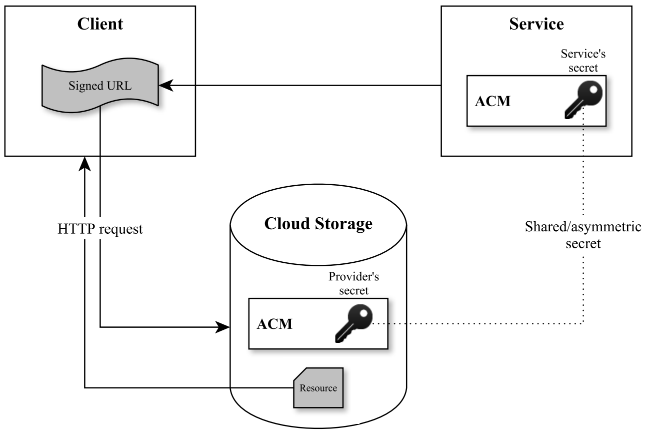


Fig. 2: The concept of Signed URLs in RESTful service architectures. The storage provider's ACM manages third party service accounts. The service implements its own ACM which manages permission of end users.

Creation:

1) A vendor specific string-to-sign S_S is assembled by concatenation of a set of request parameters P, e.g. checksum, content length, expiration time etc.
   $S_S = Concat(P)$
2) A signature Sig_S of S_S is generated by hash value calculation and application of the appropriate encryption operation, including the service's key K_S (i.e. private or shared secret key).
   $H_S = Hash(S_S)$
   $Sig_S = Enc(H_S, K_S)$
3) The URL is assembled based on the provider's HTTP API. Sig_S and P will typically be passed as query parameters or in some cases HTTP headers.

Validation:

1) The storage provider's validation service assembles a string-to-sign S_P by concatenation of P, retrieved from the request.
   $S_P = Concat(P)$
2) S_P is hashed and the service side hash is retrieved by decrypting Sig_S, using the provider's key (i.e. public or shared secret key).
   $H_P = Hash(S_P)$
   $H_S = Dec(Sig_S, K_P)$
3) The request is valid if both hashes H_P and H_S match and additional criteria (e.g. *correct content checksum*, *request not expired*) are met.

The particular implementations of the creation and validation steps depend on the CSP's contract. It has to specify a detailed concatenation procedure to assemble the required string-to-sign. Due to the nature of hash functions a single misplaced character leads to an invalid string and consequently an invalid signature. The second part of the contract is the hash function and the encryption algorithm. Since those should be based on state-of-the-art security standards, any implementation can be chosen. Finally, the structure of the URL has to be particularized by defining the addressing of resources and query parameters and headers that may be mandatory or optional.

V. Evaluation

In cloud environments any per request cost in terms of performance or money is important, since even minor increases are usually multiplied with large numbers. As described in Section I the cloud storage pricing models are based on storage and bandwidth consumed as well as number of requests. None of those values is influenced by using Signed URLs for authentication. Although, cost is not influenced performance might be a valid concern. This section evaluates the performance overhead of authentication using Signed URLs compared to the usage of Java libraries provided by the CSPs.

We expected no difference in performance of the techniques, since both require one HTTPS request per transmission which only differs in header and query parameters. Although, computation and verification of signatures could produce some overhead that should be negligible.

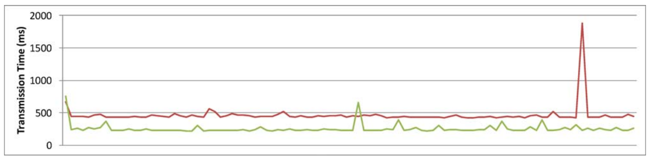
The test infrastructure, located in Germany, provided about 300 megabits downstream and 150 megabits upstream. Involved cloud storage providers are AWS, Azure and GCS. The AWS S3 container was located in the US, because European URLs are not accordingly supported in Amazon's Java SDK. Even for European targets, American hosts are used to generate URLs, which results in server side redirects back to Europe. To avoid this behavior and its potential influence on the test the default container location was used. Even though requests were naturally slower there should not have been an effect on the test results, since the target measure is the relative time increase. The Azure BLOB storage bucket was created on the region "West Europe" and the GCS container on "EU". Versions of Java libraries used are as follows:

- com.amazonaws.aws-java-sdk: 1.9.22
- com.microsoft.azure.azure-storage: 2.0.0
- com.google.appengine.tools.appengine-gcs-client: 0.4.4

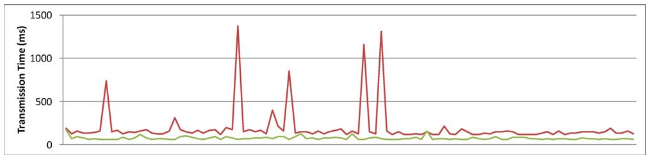
The graphs in this section shows the transmission times of 100 requests per series. Each set consists of three series, one per storage provider evaluated. On the y-axis the time in milliseconds is displayed while the x-axis represents subsequent requests. Values of *time increase* were calculated based on the relative proportion of the transmission times of Signed URLs and library usage. To balance network spikes the median of transmission times per series was used.

$$Increase = \left(\frac{MEDIAN(Series_{SignedURLs})}{MEDIAN(Series_{Library})} - 1\right) * 100\,\%$$

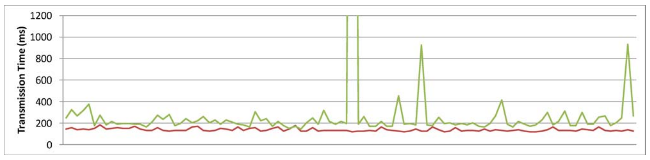
In addition to the results in this paper of 1 byte and 1 megabyte, other content sizes (1 kilobyte, 10 kilobytes, 100 kilobytes) have been investigated. The graphs were chosen to be the most representative of what we learned from the experiment.



(a) AWS S3: 88.22 % increase.

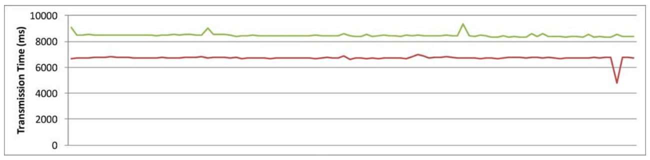


(b) Azure: 114.35 % increase.

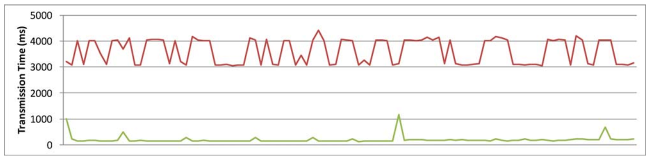


(c) GCS: 33.52 % decrease.

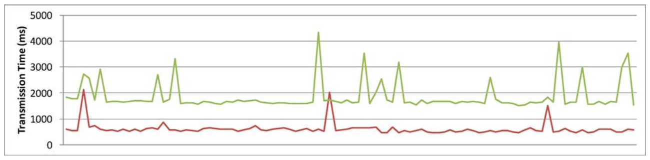
Fig. 3: Results of 100 uploads of 1 byte files comparing Signed URLs (red) and vendors' Java libraries (green) for different storage providers.



(a) AWS S3: 20.25 % decrease.



(b) Azure: 2,381.07 % increase.



(c) GCS: 65.70 % decrease.

Fig. 4: Results of 100 uploads of 1 megabyte files comparing Signed URLs (red) and vendors' Java libraries (green) for different storage providers.

A. *Comparison of Upload*

The simulation of file upload involved generation of random file contents. For each request new content was generated, hence, the content hashes were different every time. This way the utilization of potential caching mechanisms in the Java libraries was avoided. Files were not actually written to the disc, but rather handled in the computer's memory only.

The first series of tests was based on file contents of 1 byte. This test aimed to minimize the influence of file size and reveal the pure overhead of the two techniques, including the network. As Figure 3 shows, for two out of three providers Signed URLs were slower. It reveals a time increase of 88.22 % for AWS and 114.35 % for Azure. Google's Signed URLs were 33.52 % *faster*.

Figure 4 contains the results of the upload experiment with 1 megabyte files. The goal is to see the influence of file size on relation between both techniques. Figures 4a and 4b do not look as expected after the first set of tests. AWS suddenly processes Signed URLs 20.25 % faster than other requests. For Azure we can observe a drastic increase of 2,381.07 % which is basically caused by very fast regular uploads (median: 161 milliseconds). A reasonable explanation is caching and hence the avoidance of data transmission, which should not happen, since every requests contains a completely different content. For GCS the relative time decrease almost doubles to 65.70 %. In combination with other test results a trend can be observed that Signed URLs have an increased performance for bigger file sizes.

B. *Comparison of Download*

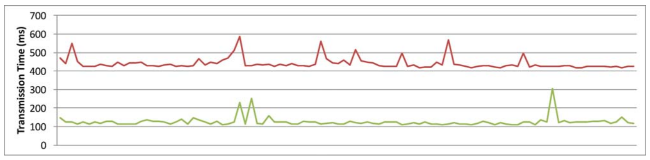
The same file contents that have been uploaded in the first sets of tests were downloaded to investigate the download behavior. Results were not stored on the disc but only buffered in memory until the termination of each download process.

Figure 5 shows downloads of 1 byte files. For all cloud storage providers an increase of transmission time for Signed URLs can be observed. On AWS it is 250.02 %, on Azure 149.32 %, and on GCS 65.56 % slower. This indicates a larger impact of signature verification overhead on the server, since the transmission effort is minimized.

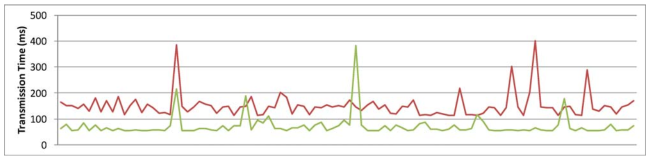
In Figure 6 a smaller increase for AWS of 16.57 % and GCS of 0.42 % but a slightly larger increase for Azure of 159.57 % can be observed. This indicates, that for Amazon and Google the overhead evens out for larger file sizes. For both providers we found the threshold for this to be true at 100 kilobytes.

VI. Discussion

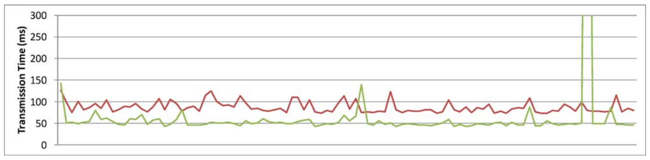
Most CSPs provide APIs to enable programmatic consumption of cloud resources, however these APIs are neither standardized nor interoperable [26], therefore inadequate for multi-cloud implementations [16]. Our experience further substantiates this assertions, our efforts to adopt the available open source multi-cloud APIs proved they were inadequate. Considering the importance of security and privacy of clients' resources in distributed systems, for us the only viable option was to implement a new approach. This complies with Cloud Security Alliance's recommendations on implementing flexible ACMs [8] that facilitate portability of cloud resources. The providers investigated in Table I support URL signing, with different specifications and implementations. The closest



(a) AWS S3: 250.02 % increase.

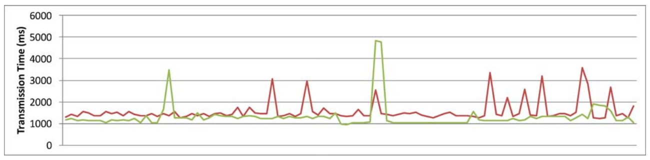


(b) Azure: 149.32 % increase.

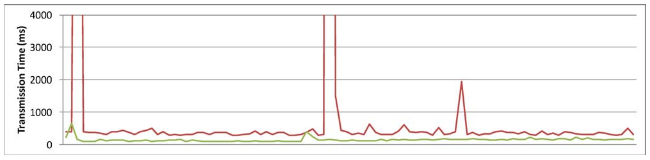


(c) GCS: 65.56 % increase.

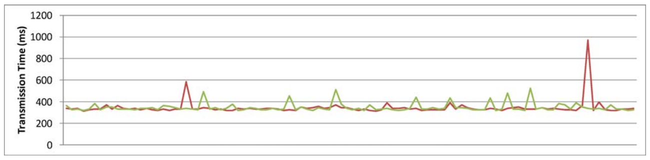
Fig. 5: Results of 100 downloads of 1 byte files comparing Signed URLs (red) and vendors' Java libraries (green) for different storage providers.



(a) AWS S3: 16.57 % increase.



(b) Azure: 159.57 % increase.



(c) GCS: 0.42 % decrease.

Fig. 6: Results of 100 downloads of 1 megabyte files comparing Signed URLs (red) and vendors' Java libraries (green) for different storage providers.

unified approach identified among them was with OpenStack compliant clouds, e.g. RackSpace and HP. Their approach also falls short of our requirements; firstly, the implementation of Signed URLs among these providers is varied. Secondly, this option restricts implementation to OpenStack cloud architecture with the implication having to still build connectors with non-compatible providers such as Google and Microsoft.

The major advantage over other authorization methods (e.g. OAuth) is the ability of programmatically issuing a security token, rather that manually allowing access based on resource requests. In particular, this means elimination of user interaction of the resource's owner to authorize other services or users. This a significant benefit in a system where the owner is represented by a software hence manual authorization is not a viable option. The following data security aspects are enhanced by integration of an ACM based on Signed URLs.

- **Data lock-in:** *In multi cloud systems users should not feel locked-in by CSPs.* A Signed URL based ACM component serves as an adapter between different cloud storage APIs. Client applications which aim to transfer stored files from one provider to another are not concerned with integrating diverse APIs, but can simply use the same HTTP methods for all providers involved. While commercial implementations may have an interest in "locking their customers in", we propose a third party system that provides standardized APIs for file transfer.
- **Data confidentiality:** In the described system files of different users are stored in the same storage with a third party being the account owner. This implies that the service owner and administrators are able to acquire full access to any data stored there. Despite a certain degree of trust users have to have in the service operator, the system design should prevent access to plain file contents. In advance of upload any file has to be encrypted with a secret known exclusively to the original file owner. The service never gets hold of this secret and consequently can not access the file content. In summary, this means data confidentiality towards other Internet users, CSPs, and the ACM service operator is secured by strong encryption.

Another very import aspect in data security is availability. At first glance, our approach has the shortcoming of decreasing a system's overall availability by involving a centralized, external ACM service. However, the service can and should be run redundantly or in a distributed way. To implement this approach, different service keys can be issued for different instances of the ACM, which is widely supported by CSPs.

From our experiments we have learned that the cloud providers behave very differently according to the file sizes processed. This is contradictory to our expectations and might be caused by limitations of storage provider's APIs, optimizations or shortcomings of Java library implementations. Therefore, software systems which support different authentication methods should be able to dynamically switch between them. E.g., very small files on AWS should be uploaded via the AWS SDK while larger files should be uploaded using Signed URLs. AWS and GCS provide shorter transmission times using Signed URLs for larger files of at least 1 megabyte in upload and equal download time for files of at least 100 kilobytes. Hence, the performance of Signed URLs can be concluded better for those two storage providers. Azure, on the other hand, provides a massively faster upload using their Java library. We assume that the library uses optimization methods, which makes the Signed URLs appear very slow in comparison.

We conclude that Signed URLs provide a base to unify cross-provider ACM and overcome the currently existing large diversity of APIs. We expect a lot of future work on the propagation of standards in cloud computing storage and service ACM as well as API design.

VII. Future Work

In our survey, we identify the Signed URLs concept as the only viable solution to implement cross-provider access control for cloud storage. Nevertheless, there is still place for improvement mainly in the standardization of the provider's ACM. The Signed URLs string concatenation, which defines the construction of the signature (see Section IV), differs between the providers. Although the community is already discussing on these issues, the establishment of a common standard could take time. In general, various provider documentations specify creation and validation of signatures in different ways. Furthermore, each provider defines disparate cryptographic hash functions, encryption algorithms, and URL construction policies.

Each Signed URL expires after a specific, pre-defined time-out. An even better solution are one-time signatures that are not yet implemented by CSPs. Hereby, each URL should only be valid for one request. If it fails, the client has to contact the ACM component again to retry.

An important aspect of cloud storage is availability. A cross-provider ACM service is often a single point of failure in a cloud-based system. One of our future tasks will be the elimination of this viability, e.g. by designing a redundant or distributed service.

References

[1] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," *Journal of Internet Services and Applications*, vol. 1, no. 1, pp. 7–18, 2010.

[2] N. Baracaldo, E. Androulaki, J. Glider, and A. Sorniotti, "Reconciling end-to-end confidentiality and data reduction in cloud storage," in *Proceedings of the 6th Edition of the ACM Workshop on Cloud Computing Security*, ser. CCSW '14. New York, NY, USA: ACM, 2014, pp. 21–32. [Online]. Available: http://doi.acm.org/10.1145/2664168.2664176

[3] The Washington Post. (2013) NSA infiltrates links to Yahoo, Google data centers worldwide, Snowden documents say. Last visited: 03-09-2014. [Online]. Available: http://www.washingtonpost.com/world/national-security/nsa-infiltrates-links-to-yahoo-google-data-centers-worldwide-snowden-documents-say/2013/10/30/e51d661e-4166-11e3-8b74-d89d714ca4dd_story.html

[4] B. T. McGeogh and B. Donnellan, "Factors that affect the adoption of cloud computing for an enterprise: A case study of cloud adoption within intel corporation," 2013. [Online]. Available: http://aisel.aisnet.org/ecis2013_cr/37/

[5] M. Vukolic, "The Byzantine empire in the intercloud," *ACM SIGACT News*, vol. 41, no. 3, pp. 105–111, 2010.

[6] M. AlZain, E. Pardede, B. Soh, and J. Thom, "Cloud Computing Security: From Single to Multi-clouds," in *System Science (HICSS), 2012 45th Hawaii International Conference on*, Jan 2012, pp. 5490–5499.

[7] S. T. Graham and X. Liu, "Critical evaluation on jclouds and cloudify abstract apis against ec2, azure and hp-cloud." in *COMPSAC Workshops*, 2014, pp. 510–515.

[8] CSA, *SecaaS Implementation Guidance Category 1 - Identity and Access Management*, Cloud Security Alliance, September 2012. [Online]. Available: https://downloads.cloudsecurityalliance.org/initiatives/secaas/SecaaS_Cat_1_IAM_Implementation_Guidance.pdf

[9] M. Schnjakin, T. Metzke, and C. Meinel, "Applying Erasure Codes for Fault Tolerance in Cloud-RAID," *IEEE 16th International Conference on Computational Science and Engineering (CSE)*, pp. 66–75, 2013.

[10] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the Clouds: A Berkeley View of Cloud Computing," *Technical Report UCB/EECS-2009, EECS Department, University of California, Berkeley*, 2009.

[11] J. Broberg, R. Buyya, and Z. Tari, "Creating a 'Cloud Storage' Mashup for High Performance, Low Cost Content Delivery," *Service-Oriented Computing (Volume 5472), ICSOC'08 Workshops*, April 2009.

[12] H. Abu-Libdeh, L. Princehouse, and H. Weatherspoon, "RACS: A Case for Cloud Storage Diversity," *SoCC'10*, June 2010.

[13] H. C. H. Chen, Y. Hu, P. P. C. Lee, and Y. Tang, "NCCloud: A Network-Coding-Based Storage System in a Cloud-of-Clouds." *IEEE Trans. Computers*, pp. 31–44, 2014.

[14] A. Bessani, M. Correia, B. Quaresma, F. André, and P. Sousa, "DepSky: dependable and secure storage in a cloud-of-clouds," in *Proceedings of the sixth conference on Computer systems*, ser. EuroSys '11. New York, NY, USA: ACM, 2011, pp. 31–46. [Online]. Available: http://doi.acm.org/10.1145/1966445.1966449

[15] C. S. C. Council, "Cloud security standards:what to expect & what to negotiate," October 2013. [Online]. Available: http://www.cloud-council.org/Cloud_Security_Standards_Landscape_Final.pdf

[16] D. Petcu, C. Craciun, and M. Rak, "Towards a Cross Platform Cloud API - Components for Cloud Federation," in *CLOSER*, 2011, pp. 166–169.

[17] T. Reimer, P. Abraham, and Q. Tan, "Federated identity access broker pattern for cloud computing," in *16th International Conference on Network-Based Information Systems*, 2013.

[18] Google, "Google Cloud Storage - A simple way to store, protect, and share data," *Google Whitepaper*, 2012.

[19] ——. (2014) Google Cloud Storage Access Control. Last visited: 04-09-2014. [Online]. Available: https://developers.google.com/storage/docs/accesscontrol

[20] P. Mehner, *Developing Cloud Applications with Windows Azure Storage*, 1st ed. Microsoft Press, March 2013, ISBN: 978-0735667983.

[21] Amazon, *Amazon Simple Storage Service - Developer Guide*, Amazon Web Services, 2014, API version: 2006-03-01.

[22] ——. (2014) Introduction to Managing Access Permissions to Your Amazon S3 Resources. Last visited: 05-09-2014. [Online]. Available: http://docs.aws.amazon.com/AmazonS3/latest/dev/intro-managing-access-s3-resources.html

[23] OpenStack, *OpenStack Object Storage Administrator Guide*, OpenStack, August 2014.

[24] D. Chadwick, K. Siu, C. Lee, Y. Fouillat, and D. Germonville, "Adding Federated Identity Management to OpenStack," *Journal of Grid Computing*, vol. 12, no. 1, pp. 3–27, 2014. [Online]. Available: http://dx.doi.org/10.1007/s10723-013-9283-2

[25] Internet Engineering Task Force (IETF). (2012) The OAuth 2.0 Authorization Framework. Last visited: 05-09-2014. [Online]. Available: http://www.rfc-base.org/rfc-6749.html

[26] N. Loutas, E. Kamateri, F. Bosi, and K. Tarabanis, "Cloud Computing Interoperability: The State of Play," in *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, Nov 2011, pp. 752–757.