

# Secure Access Mechanism for Cloud Storage

Danny Harnik, Elliot K. Kolodner, Shahar Ronen, Julian Satran, Alexandra Shulman-Peleg, Sivan Tal

IBM Haifa Research Lab

Haifa, Israel 31905

email: {dannyh, kolodner, shaharr, julian\_satran, shulmana, sivant}@il.ibm.com

**Abstract**—Emerging storage cloud systems provide continuously available and highly scalable storage services to millions of geographically distributed clients. A secure access control mechanism is a crucial prerequisite for allowing clients to entrust their data to such cloud services. The seamlessly unlimited scale of the cloud and the new usage scenarios that accompany it pose new challenges in the design of such access control systems.

In this paper we present a capability-based access control model and architecture appropriate for cloud storage systems that is secure, flexible, and scalable. We introduce new functionalities such as a flexible and dynamic description of resources; an advanced delegation mechanism and support for auditability, accountability and access confinement. The paper details the secure access model, shows how it fits in a scalable storage cloud architecture, and analyzes its security and performance.

## 1 INTRODUCTION

The rapid growth in the amount of personal and organizational digital data is a major characteristic of information systems in this decade. This growth is accompanied by increased demand for availability, as users wish to run their software and access their data, regardless of their type and location, from any web-enabled device at any time. Cloud computing addresses these challenges by providing virtualized resources as an online service and by allowing them to scale dynamically. Services offered over the cloud include applications (Software as a Service), virtual machines (Infrastructure as a Service), and data storage (Storage as a Service). The latter service, along with the storage cloud upon which it resides, is the topic of this paper.

Storage cloud systems are required to provide continuous availability and elastic scalability [2] while serving *multiple tenants* accessing their data *through the Internet*. The storage cloud infrastructures are comprised of tens of geographically dispersed data centers (DCs), where each DC is comprised of many thousands of compute and storage nodes, and should be able to efficiently serve millions of clients sharing billions of objects. Furthermore, to achieve availability and scalability, each data object is replicated at multiple data centers across the cloud. Each object replica should be accessible for reads and writes, and to optimize access latency and data throughput, it is preferable that clients be directed to the replica closest to them.

In addition to the scale and availability requirements, today's new web applications such as mashups, introduce new data access practices. Namely, data is not necessarily accessed directly by its original owner but rather through various applications, in flexible sharing scenarios and with various

billing methods. These applications put forth the following functional requirements for the access control mechanisms.

*Secure chaining of services:* When a client uses a social networking application, this application often relies on another service to perform an action, which in turn may invoke yet another application and so on. When chaining services in this way, it is important to keep track of the access rights of the client at the end of the chain; otherwise, the system could become vulnerable to attacks like clickjacking and cross-site request forgery that can occur when accessing the web [3]. Such attacks can be prevented by giving each client the least authority required to perform its request, and correctly maintaining the access permissions of the client along the chain, i.e., taking care that they are neither expanded nor restricted.

Most of the existing commercial cloud storage systems [1], [4], [15], [19] use *Access Control Lists (ACLs)* for authorization, which is done in the data access path. This approach has limitations in its ability to chain services without losing track of the privileges of the principals, thus failing to enforce the principle of least privilege (as described in [3]). Furthermore, every data access request in these systems is authenticated using an authentication string based on a secret key that the user shares with the storage system. Thus, using chains of services requires moving the control of the data from the end user to the service provider accessing the data on his behalf.

*User-to-user and user-to-application access delegation:* The cloud systems should provide a convenient infrastructure supporting user-to-user and user-to-application access delegation across different social networks. For example, an application like Facebook, which is granted access rights to a certain user repository should be able to further delegate the rights to other users, according to the policy defined by the owner (e.g. delegation to a certain friend). Thus, the delegation should be transitive and fine-grained [17], allowing a user to delegate a subset of rights to a subset of resources (without the need to give one service full access to all the data managed by another service like it is done today). Finally, in order to provide a secure environment that allows auditing and compliance to regulations, it should always be possible to determine who had access to a specific resource and who is responsible for the delegation.

While some experimental distributed file systems and file-sharing services meet the user-to-user delegation requirements (e.g., Fileteller [10], DisCFS [16] and WebDAVA [14]), they are based on the Public Key Infrastructure (PKI), which has

been shown to have multiple limitations for adoption in cloud-scale systems [20]. On the other hand, efforts like OAuth[8] which do not assume PKI, require that access delegation grants pass through the service provider, thus not meeting our user-to-user delegation requirement, and also imposing a communication overhead.

Existing cloud systems, like Amazon S3 [1] or EMC Atmos [4] allow a limited user-to-user delegation in which an authorized user creates an HTTP request with embedded authentication information and passes it to another user or web application. This form of access delegation is very limited as it requires repetitive 3-way interaction for every access request, with two phases of authentication and authorization - one between the granter of the access and the grantee of the access, and one between the granter and storage system. Microsoft Windows Azure Storage [15] supports a limited capability-based access using a *Shared Access Signature*, which encodes a capability to perform specified operations on a specified resource for a specified period of time. This mechanism is an advance, but it is limited in scope and has to be managed by the user. Furthermore, it suffers from replay attack vulnerability, and has issues with auditability and accountability.

Finally, all of the existing commercial cloud storage services limit the definition of resources for delegation to fixed criteria and do not support dynamic criteria (e.g., all objects created after a certain date), which are important for advanced applications that handle large number of objects.

*Scalability and high availability:* Cloud services are expected to always be available, even during failures and disasters, and to adjust themselves to significant load fluctuations within short periods of time. Distribution of the access control system is a key instrument in meeting these goals, without creating bottlenecks, or single points of failure that can also become an attractive attack target (e.g. [20]). Any point in the cloud possessing a resource should be able to serve the request while enforcing access control and the access control mechanism must not substantially affect the data access latency. Most commercial cloud storage systems perform authentication and authorization on the data access path, when serving each request. This imposes limitations on the system scalability and availability and creates new bottlenecks at identification or key management servers.

Some existing capability-based models, for example, OSD [5], [18], [13], SCARED [22], and CbCS [6], satisfy the requirements of secure chaining of services and user-to-user delegation. However, these protocols are not geared toward the scalability and high availability required by the cloud environment.

In summary, today's cloud solutions fall short of satisfying the requirements we define above. In this paper we propose a capability-based access control model designed to address the emerging challenges and data models of storage clouds, allowing relocation of social networks storage to cloud repositories. Extending previous capability-based access control models [22], [5], [13], [6] we introduce the following innovations:

- We present a model supporting fine grain and dynamic

scope for access rights. We introduce a dynamic definition of the resources via a flexible description with pattern matching on object attributes (e.g. object name, type, or metadata). Thus, the accessible subset of resources can change as objects are added or removed from the system.

- We enable user-to-user and user-to-application delegation, where one user can directly delegate a subset of his access rights to a subset of resources without requiring the intervention of a system administrator or the storage cloud. We introduce mechanisms for auditing and access confinement that can serve as a basis for determining accountability, compliance and billing.
- We present an architecture of a data center that supports capability-based access control. Our system is scalable and allows integrating external identity or access management components used by existing social networking applications. It separates authentication and authorization from the data access path, allowing non-mediated and efficient access to data without the need to specify the accessed storage server (i.e. the same credential is valid for all the replicas regardless of their physical location).

Figure 1 illustrates the advantages of the model for social networking applications through a scenario in which Alice uploads her photos to a cloud storage repository which she owns. Then, she delegates access to photos matching a certain description to her social networking application. According to the policy defined by Alice this service gives her friend Bob the permissions to add his photos. When Alice delegates access to these photos (including Bob's) to another photo sharing application, they are immediately available without the need to upload or download the data.

## 2 SECURE ACCESS PROTOCOL

In this section we present a capability-based access control mechanism for secure access to objects in a storage cloud. We assume the simple and most common cloud storage data model [1], [4], which is comprised of the following types of resources: **Namespaces**, which are abstract containers providing context (such as ownership) for the data objects; and **Data objects** containing arbitrary content and having a unique identifier within the namespace they belong to.

Capability-based access protocols involve three entities [5]: *clients*; a *security/policy manager*, which authenticates and authorizes the clients and grants the credentials; and *storage servers* that enforce access control by validating the credentials. The access flow in such systems consists of two stages: (1) the client requests a credential from the security manager and then (2) uses it for accessing the data on the storage servers. The *enforcement point* is the component in a storage server that enforces access control.

### 2.1 Capabilities

A *capability* is a data structure that encodes certain access rights on one or more identified resources. Below we describe the capability fields enabling the new functionalities.

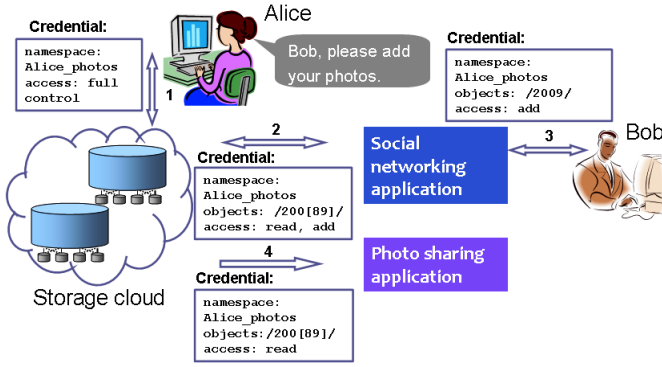


Fig. 1. **Delegation chaining.** Examples of user-to-user and user-to-application delegation: (1) Alice creates a personal repository in a storage cloud. She has full control over her namespace named *Alice\_photos*. (2) Alice delegates access to photos containing "2008" or "2009" in their title to her social networking application (granting read and add permission). (3) According to a policy defined by Alice the social networking service further delegates access to Bob, allowing him to add photos to the namespace *Alice\_photos*. Note that there are two options to generate the credential granted to Bob. It can be either created by Alice or generated by the social networking application. (4) Alice delegates access to her photo sharing application by generating a credential with the read access to photos matching the pattern */200[89]/*, which include the photos uploaded by Bob as well.

**Flexible and dynamic capability scope:** In our model, the capabilities can provide a flexible description of resources based on attributes such as object name, type and metadata. To allow this, the *ResourceDescriptor* component in the capability can contain regular expressions and other selection criteria. This enables efficient data access management by allowing creation of capabilities with dynamic scope describing resources that can change over time (e.g., as new objects matching the selection criteria are added). For example, a "housekeeping" application may receive a capability to move, delete or compress objects that are older than a specified age, or have not been accessed for a specified amount of time.

**Flexible access rights:** The *Permissions* field in the capability can specify any type of operation which is properly defined to the security manager and the enforcement point. For example, in addition to standard operations such as *Create*, *Read*, *Update*, *Delete*, or even *UpdateMetadata*, it can specify operations exclusive to specific content types such as *Compress*, *Resize*, *RotateFlip* or *ReduceResolution*. Such operations are introduced in some storage cloud offerings (e.g., Nirvanix[19] already added methods like *RotateFlip*). Using our proposed protocol, the granularity of the access policy can match the functionality supported by the storage service.

## 2.2 Credentials and access control flow

Below we describe the two stages of capability-based access suited for the cloud storage data model.

**Obtaining the credentials:** A client sends a request for a capability to a security manager, which in turn validates the identity of the requestor and performs an authorization

process. If the request is approved, the security manager responds with the credential comprised of the *capability* and the *capability-key*. The capability (*CAP*) denotes the public part of the credential specifying one or more resources and the granted access rights. The capability-key (*CAP\_KEY*) is the cryptographic hardening of a capability with a secret key and pseudorandom function (*PRF*), which can be a keyed cryptographic hash such as HMAC-SHA1 or HMAC-SHA256. The key used for this operation, *KeyNS*, is shared by the security manager and the storage server, for the addressed namespace. Since the capability-key is the secret part of the credential it should be sent to the client in encrypted form, either as part of the protocol or by using an encrypted communication channel.

$$\begin{aligned} \text{Credential} &= [\text{CAP}, \text{CAP\_KEY}] \\ \text{CAP\_KEY} &= \text{PRF}_{\text{KeyNS}}(\text{CAP}) \end{aligned}$$

**Accessing the resource.:** To access the object of interest the client attaches to the *Request* (typically a combination of a method and data, e.g., *write* and the contents of the object to be written), a credential consisting of two parts: the capability and the validation tag (*Val\_Tag*). The validation tag is a keyed hash that is computed with the client's capability-key; it is used to authenticate the capability and additional information as defined by the security method described below. The parameter used for the *token* depends on the security method that is encoded in the capability's *SecurityInfo* field.

$$\begin{aligned} \text{Message} &= [\text{Request}, \text{CAP}, \text{Val\_Tag}] \\ \text{Val\_Tag} &= \text{PRF}_{\text{CAP\_KEY}}(\text{token}) \end{aligned}$$

The choice of the security method depends on the guarantees of the underlying communication channel. Below are two security methods that are suitable for the secure and insecure channels respectively:

- **Channel identifier (CHID)** This method is similar to the CAPKEY method of the OSD protocol [5]. It is suitable for use over a protected communication channel such as IPSEC or HTTPS. In this case the *Token* is taken to be the unique channel identifier of the protected communication channel. Since the validation tag binds the messages to the original channel, an eavesdropper cannot modify messages on the channel, replay messages on it or initiate new messages.
- **Message headers (MSGH).** Suitable for access via an open HTTP protocol. In this case the *token* contains some of the HTTP message fields that are significant in terms of access rights. These include the standard HTTP headers such as the HTTP method and resource URI, as well as Host, Date, Content-Type and Content-MD5. Including a content-MD5 field in the generation of the validation tag guarantees that the data cannot be modified and an eavesdropper intercepting the message can only use it to repeat the exact same command, in the same time frame as specified.

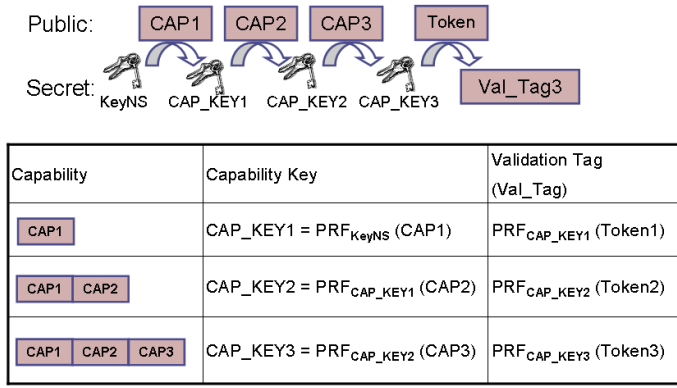


Fig. 2. **Delegation chaining.** Access delegation among 3 different users. The security manager uses the namespace key,  $KeyNS$ , to generate a capability,  $CAP1$ , and a capability key  $CAP\_KEY1$  for the first user. Using this key the user can create validation tags for his requests as well as generate the credentials for the second user, who in turn can generate the credentials for the third.

When a request message is received at the enforcement point in the storage server, it is validated by performing the following steps:

- *Calculating the capability-key.* The capability-key is computed based on the namespace key shared with the security manager and the received capability.
- *Calculating the expected validation tag.* The capability-key computed in the previous step is used to compute the expected validation tag, according to the security method specified in the capability.
- *Comparing the validation tags.* The received validation tag is compared to the expected validation tag.
- *Validating the capability and request.* The received capability is validated to be formatted correctly and to match the requested operation.

### 2.3 User-to-user delegation

In this section we describe an advanced delegation mechanism that enables a user to create (and pass to another user) a new capability that encodes a subset of the rights on a subset of the resources encoded in an existing capability. We leverage and enhance a delegation technique of Reed et al. [22] and adapt it for the requirements of storage clouds and Web 2.0 applications.

**Delegation mechanism:** Having a credential  $[CAP, CAP\_KEY]$ , a *reduced-capability* credential can be created by appending a reduced capability to the original capability, and creating a new capability-key by hashing the appended part using the original capability-key as the key.

Formally, a delegation chain is created as follows:

$$\begin{aligned}
 Credential_1 &= [CAP_1, PRF_{KeyNS}(CAP_1)] \\
 \text{For } n > 1 : \\
 CAP\_KEY_n &= PRF_{CAP\_KEY_{n-1}}(CAP_n) \\
 Credential_n &= [(CAP_1, \dots, CAP_n), CAP\_KEY_n]
 \end{aligned}$$

Notes:

- A client cannot generate  $Credential_1$  because that requires knowledge of  $KeyNS$ .
- Generating  $CAP\_KEY_n$  requires knowledge of  $CAP\_KEY_{n-1}$ . Yet,  $CAP\_KEY_n$  does not reveal any useful information regarding  $CAP\_KEY_{n-1}$ .
- It is crucial that all capability keys be sent over authenticated and encrypted channels.
- Our key generation technique computes  $CAP\_KEY_n$  based only on  $CAP_n$  as opposed to  $(CAP_1, \dots, CAP_n)$  used by Reed et al. [22]. This reduces the cryptographic overhead, while preserving the same level of security.

This mechanism of creating new capabilities and capability-keys based on existing ones is illustrated in Figure 2.

**Reduction of access rights:** When allowing user-to-user delegation it is important to ensure that each user can only reduce his own access rights and cannot delegate access beyond his own permission. We define one capability  $CAP_{i+1}$  to be a subset of another capability  $CAP_i$  if and only if the value of each field in  $CAP_{i+1}$  does not describe a higher privilege than the corresponding field in  $CAP_i$ . For example, we require that during delegation the new *ResourceDescriptor* describes a subset of the resources, and *Permissions* describes a subset of the operations, and the *ExpiryTime* field contains a lower or equal expiration time. Two fields are an exception to this rule: (1) We require that the *SecurityInfo* of the two capabilities contain exactly the same security method; (2) we do not compare the *Discriminator* fields.

**Delegation validation:** When a chained capability credential is processed at the enforcement point it is validated by the following mechanism, which is an extension of the one described above.

- *Calculation of the chain of capability keys.* First,  $CAP\_KEY_1$  is calculated based on  $CAP_1$  and  $KeyNS$ . Then, each subsequent  $CAP\_KEY_i$  is calculated from  $CAP\_KEY_{i-1}$  and  $CAP_i$  as described above (see Figure 2).
- *Calculation of the validation tag.* The last capability-key in the chain is used to calculate the expected validation tag.
- *Compare the validation tags.* Compare the received vs. the expected tags.
- *Validation of the capability delegation chain and the request.* Validate that each capability in the chain is a proper subset of the preceding capability as defined above and that the request conforms with the last capability. This ensures one cannot increase the access granted by a given credential.

### 2.4 Identity Testing and Access Confinement

Sometimes it may be desirable to limit the use of a capability granted to a specific user or group as an extra measure of security, typically for more sensitive data. One consideration is that one may grant a user access to an object, but not trust him to keep his capability credential securely without leaking

it. For this purpose, we add an optional *Identity* field in the capability that can be used to confine the credential and make it usable only by the specified identity. It may contain the identity of a principal or a group. The *Identity* is filled in by the delegating person and is verified by the enforcement point. Using this field allows enforcing the *access confinement* property, defined by Lampson [12].

For credentials in which the *Identity* field is used, the enforcement point needs to authenticate the requesting client's identity, and potentially validate his membership in a group. To speed up this process caching of relevant identity information at the enforcement point may be implemented [11]. It must be noted that the authorization is still separated from the data path.

### 2.5 Auditability and accountability

In most cases, an access control system is required to provide mechanisms for auditing and accountability of all accesses to a resource. Auditing is of special concern in capability-based access control systems, since a capability-based credential, once obtained, can be used anonymously without having to authenticate the requester's identity. To address this concern, an *Audit* field is added to the capability. For instance, the security manager can store in this field the identity of the client to which the capability was issued. Since the *Audit* field is part of the capability, it is authenticated by the capability-key and cannot be modified. The enforcement point can log this information so it can later be used to learn which capabilities were used and when. This can serve as a tool for *billing* in the cloud, as it may record who is accountable for each access, and is especially useful in implementing today's new pay-per-access policies.

### 2.6 Delegation Security

A central feature in our protocol is the extensive delegation capability. The main concern is the fact that in delegation, capabilities can actually be created by an external principal (external to the cloud) rather than the security manager component. Nevertheless, this additional feature does not compromise the overall security of the protocol. An attack on the system consists of a malicious party performing an operation that it is not allowed to do. For this to happen it must produce a series of capabilities along with a corresponding validation tag for its request. There are basically two options for an adversary to present such a series. The first option is to generate a validation tag without knowledge of the end capability key, which he is incapable of doing. The other option is that somewhere, along the chain of delegation, the adversary obtained a capability key  $CAP\_KEY_i$  for a capability of its choice without knowledge of the previous key in the chain  $CAP\_KEY_{i-1}$ . This too is ruled out by the properties of the underlying pseudorandom function.

## 3 ARCHITECTURE AND IMPLEMENTATION

We developed an architecture of a storage cloud that supports the presented capability-based model, while satisfying

the scalability and high availability requirements. We consider a storage cloud comprised of a set of geographically dispersed data centers (DCs), collectively storing billions of objects and peta-bytes of storage capacity. Each DC stores and manages a subset of the cloud's resources and there is no single component that contains all the information about all the objects in the storage cloud. In our solution, the same access credential is valid for retrieving all the replicas of an object stored at different data centers. Below we detail the main components of our access control architecture.

- *The Enforcement Point* processes all data access requests and validates the client requests and credentials. Every server/node in the cluster holds an instance of this component.
- *The Security Manager* is responsible for handling authorization requests and generating credentials. It communicates with the Identity Manager and the Access Manager components to authenticate the user and authorize his request for a credential. Using the key generated by the Key Manager, the Security Manager generates the capability and capability-key, which are returned to the client. It is also responsible for the key exchange with the Enforcement Point, allowing using the same access credentials to all the replicas of an object. Each Security Manager is responsible for a certain subset of resources, which usually reside in the same DC.
- *The Identity Manager* authentication component responsible for verifying the identity of a principal requesting authorization, as well as group membership inquiries. Under the orchestration of the Security Manager, multiple external identity managers, that are not part of the data center, can be deployed in the cloud. For example, some social networking applications may wish to integrate their own identity management servers. In the scenario illustrated in Figure 1 this will allow identification of Bob as well as access confinement as described in Section 2.4. On the other hand, many Web 2.0 and Mashup applications may rely on the identity management of a public cloud and use the credentials generated for the cloud clients (e.g., using a credential generated for Alice by the cloud provider).
- *The Access Manager* responsible for authorization and access policy management. A scalable design for this component distributes the access policy information across the data centers (e.g., in our implementation each DC stores the policy information for the namespaces in its responsibility). Similarly to identity managers, the cloud architecture allows deployment of multiple external access managers. This allows the social applications to maintain their own access policies, while relying on the cloud security manager as the credential generation authority. However, this is not mandatory and applications may select to maintain internal access managers that will be responsible for the delegation of credentials generated by the cloud provider.

```

GET SPI/report-March-2009.doc HTTP/1.1
host: www.cloud.acme.com
x-acme-credential:
  [{ capability : [
    { ResourceDescriptor :
      [{ nsid : SPI }, { security tag : 1 } ]
      [{ oid = "/"^report.+200[89]$/ } ]
    { Operations : read, add },
    { Discriminator : 0xFDCED0016EE87953472 },
    { Identity : }, { Audit : Bob },
    { ExpiryTime : "Thu, 31 Jan 2011 17:15:03 GMT },
    { Delegatability : 0 } ]
  { Val_Tag : 0xAABF099916EE87953472 } ] ]

```

Fig. 3. HTTP request to GET an object from the storage cloud. A credential containing a capability and a validation tag are added to the HTTP header. The credential value is represented as a JSON string.

- *The Key Manager* is the key management component responsible for the secure generation, serving and storage of the cryptographic keys shared between the Security Manager and the Enforcement Point. Every namespace has a set of keys associated with it, and efficient key management is provided by storing a copy of the namespace keys in each DC that contains a replica of that namespace.

### 3.1 RESTful Implementation

We developed a prototype of a storage cloud comprised of data centers with the architecture described above. One of our design principles was to support the philosophy of REST (REpresentational State Transfer) [7], which became widely adopted due to its simplicity, scalability and easy deployment [21], [9]. In order to implement the capability-based access control model as part of a RESTful web service, we followed all the fundamental principles of REST, which in our set up can be formulated as follows: (1) All resources, including objects, namespaces and credentials are referenced using a uniform resource identifier (URI); (2) All resources are manipulated by the basic HTTP methods (GET, PUT, DELETE and POST); and (3) All resource requests are stateless.

To support this, our system provides a RESTful API for the communication with the Security Manager, addressing the requested credential as a resource. In essence, an authorization request is sent using an HTTP GET request to a resource of type *credential*. To allow the easy integration of credentials in data access requests, we embed them in the HTTP headers of namespace and data object requests. They are represented as JavaScript Object Notation (JSON) strings, as illustrated in Figure 3, which presents an example of a user request to access (GET) an object. It shows the capability and the validation tag that are sent in an HTTP request of a client to the server. Our prototype shows that it is possible to implement the capability-based access control model as a RESTful protocol in which all the resources (including the credentials) are controlled by components that communicate via a standardized HTTP interface and exchange representations of these resources.

## 4 CONCLUSIONS AND FUTURE WORK

We present an access control mechanism, which addresses the new challenges brought forth by the scale and applications

introduced in the cloud setting. Observing that most ACL-based systems are limited in their ability to support such features, we developed a capability-based system that addresses them. Furthermore, we present a general architecture of a data center in a storage cloud with integrated security components that addresses the scalability requirements of storage cloud systems. We built a prototype implementing each of the components.

In the future we intend to work on the integration of this architecture with existing enterprise systems. We hope to use our experimental system to evaluate the overall performance of the access control mechanism on real workloads. We are encouraged by our initial evaluation and are working to incorporate this work in a production storage cloud system. Lastly, we aim to address the challenges raised in federation scenarios, where several enterprises need to federate their resources across geographically distributed administrative domains, while ensuring comprehensive and transparent data interoperability.

## 5 ACKNOWLEDGMENTS

We thank Dalit Naor for her contribution to the presented ideas. We thank Guy Laden, Eran Rom and Aviad Zuck for contribution to the development of the data center architecture. The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n 257019.

## REFERENCES

- [1] *Amazon Simple Storage Service (Amazon S3)*, Amazon, <http://aws.amazon.com/s3/>.
- [2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the clouds: A Berkeley view of cloud computing," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28, February 2009, <http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html>.
- [3] T. Close, "ACLs don't," <http://www.hpl.hp.com/techreports/2009/HPL-2009-20.pdf>.
- [4] *Atmos Online Programmer's Guide*, EMC, <https://community.emc.com/docs/DOC-3481>, accessed Jan 12, 2010.
- [5] M. Factor, D. Nagle, D. Naor, E. Riedel, and J. Satran, "The OSD security protocol," in *IEEE Security in Storage Workshop*, 2005, pp. 29–39.
- [6] M. Factor, D. Naor, E. Rom, J. Satran, and S. Tal, "Capability based secure access control to networked storage devices," in *MSST '07: Proceedings of the 24th IEEE Conference on Mass Storage Systems and Technologies*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 114–128.
- [7] R. T. Fielding and R. N. Taylor, "Principled design of the modern web architecture," *ACM Trans. Internet Technol.*, vol. 2, no. 2, pp. 115–150, May 2002. [Online]. Available: <http://dx.doi.org/10.1145/514183.514185>
- [8] E. Hammer-Lahav, *The OAuth 1.0 Protocol*, Internet Engineering Task Force, February 2010, <http://tools.ietf.org/html/draft-hammer-oauth-10>.
- [9] H. Han, S. Kim, H. Jung, H. Y. Yeom, C. Yoon, J. Park, and Y. Lee, "A RESTful approach to the management of cloud infrastructure," in *Proceedings of the 2009 IEEE International Conference on Cloud Computing*, ser. CLOUD '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 139–142. [Online]. Available: <http://dx.doi.org/10.1109/CLOUD.2009.68>
- [10] J. Ioannidis, S. Ioannidis, A. D. Keromytis, and V. Prevelakis, "Fileteller: Paying and getting paid for file storage," in *Financial Cryptography*, ser. Lecture Notes in Computer Science, M. Blaze, Ed., vol. 2357. Springer, 2002, pp. 282–299.

- [11] P. Karger, "Improving security and performance for capability systems, ph.d. dissertation," Cambridge, England, Tech. Rep. 149, 1988.
- [12] B. Lampson, "A note on the confinement problem," *Commun. ACM*, vol. 16, no. 10, pp. 613–615, 1973.
- [13] A. W. Leung, E. L. Miller, and S. Jones, "Scalable security for petascale parallel file systems," in *SC '07: Proceedings of the 2007 ACM/IEEE conference on Supercomputing*. New York, NY, USA: ACM, 2007, pp. 1–12.
- [14] A. Levine, V. Prevelakis, J. Ioannidis, S. Ioannidis, and A. D. Keromytis, "WebDAVA: An administrator-free approach to web file-sharing," in *WETICE*. IEEE Computer Society, 2003, pp. 59–64.
- [15] *Windows Azure Platform*, Microsoft, <http://www.microsoft.com/windowsazure/windowsazure/>.
- [16] S. Miltchev, V. Prevelakis, S. Ioannidis, J. Ioannidis, A. D. Keromytis, and J. M. Smith, "Secure and flexible global file sharing," in *USENIX Annual Technical Conference, FREENIX Track*. USENIX, 2003, pp. 165–178.
- [17] S. Miltchev, J. M. Smith, V. Prevelakis, A. Keromytis, and S. Ioannidis, "Decentralized access control in distributed file systems," *ACM Comput. Surv.*, vol. 40, no. 3, pp. 1–30, 2008.
- [18] D. Nagle, M. Factor, S. Iren, D. Naor, E. Riedel, O. Rodeh, and J. Satran, "The ANSI T10 object-based storage standard and current implementations," *IBM Journal of Research and Development*, vol. 52, no. 4-5, pp. 401–412, 2008.
- [19] *Nirvanix Web Services API Developer's Guide*, Nirvanix, <http://developer.nirvanix.com/sitefiles/1000/API.html>, accessed Jan 13, 2010.
- [20] Z. Niu, H. Jiang, K. Zhou, T. Yang, and W. Yan, "Identification and authentication in large-scale storage systems," *Networking, Architecture, and Storage, International Conference on*, vol. 0, pp. 421–427, 2009.
- [21] C. Pautasso, O. Zimmermann, and F. Leymann, "Restful web services vs. "big" web services: making the right architectural decision," in *Proceeding of the 17th international conference on World Wide Web*, ser. WWW '08. New York, NY, USA: ACM, 2008, pp. 805–814. [Online]. Available: <http://doi.acm.org/10.1145/1367497.1367606>
- [22] B. C. Reed, E. G. Chron, R. C. Burns, and D. D. Long, "Authenticating network-attached storage," *IEEE Micro*, vol. 20, pp. 49–57, 2000.