In proceedings of IEEE International Conference on Cloud Computing (CLOUD), 2016

Managing Cloud Storage Obliviously

Vaishali Narkhede, Karuna Joshi and Tim Finin

Department of Computer Science and Electrical Engineering

University of Maryland, Baltimore County

Baltimore, MD 21227, USA

{vaishal1, karuna.joshi, finin}@umbc.edu

Adam J. Aviv, Seung Geol Choi and Daniel S. Roche

Department of Computer Science

United States Naval Academy

Annapolis, MD 21402, USA

{aviv, choi, roche}@usna.edu

***Abstract* - Consumers want to ensure that their enterprise data is stored securely and obliviously on the cloud, such that the data objects or their access patterns are not revealed to anyone, including the cloud provider, in the public cloud environment. We have created a detailed ontology describing the oblivious cloud storage models and role based access controls that should be in place to manage this risk. We have developed an algorithm to store cloud data using oblivious data structure defined in this paper. We have also implemented the *ObliviCloudManager* application that allows users to manage their cloud data by validating it before storing it in an oblivious data structure. Our application uses role-based access control model and collection based document management to store and retrieve data efficiently. Cloud consumers can use our system to define policies for storing data obliviously and manage storage on untrusted cloud platforms even if they are unfamiliar with the underlying technology and concepts of oblivious data structures.**

Keywords— Cloud Computing, cloud security, oblivious storage, ontologies, oblivious data structure.

I. INTRODUCTION

Public cloud infrastructure is attracting many organizations to migrate their enterprise data to the cloud, for reasons such as cost savings, maintenance, and scalability. However, there are security and privacy concerns [21] to be addressed before customers readily outsource their sensitive data. To be effective, a cloud storage service should provide, at a minimum, features and security guarantees like confidentiality, integrity, availability, reliability, efficient retrieval and data sharing [32]. Most cloud consumers want their cloud data and its usage to remain private, so along with encryption of their data objects, they also want their access patterns to be oblivious to the cloud provider. Various approaches like Oblivious RAM (ORAM) [1][2] and Oblivious Storage [23] have been developed to address obliviousness of data.

This work makes three key contributions. First, we have created a detailed ontology describing the oblivious cloud storage models, security policies and role-based access controls that should be in place to manage risk. Secondly, we have developed an algorithm to store cloud data using a variable ORAM approach [21]. Lastly, we have implemented an *ObliviCloud-Manager* application that allows users to manage their cloud data by validating it before storing it in an oblivious data structure. In this paper we briefly describe the ontology and the algorithm, and focus mainly on the ObliviCloudManager application that is web-based and can be hosted on customer's trusted infrastructure.

The oblivious map data structure system provides a strong privacy guarantee such that neither the raw network-traffic data nor the access pattern on the server reveal information about the customer's data. To use this system, cloud consumers need only specify the security configuration and desired level of security to enforce in order to store their data securely on the cloud.

II. RELATED WORK

We have reviewed literature on various oblivious data structures available for high level privacy in cloud storage [1-13]. We have also taken into consideration the National Institute of Standards and Technology (NIST) [7, 8] and Cloud Security Alliance (CSA) [16] guidelines for secure cloud storage and ontologies that define cloud security and compliance. For secure deletion of cloud data, we have considered the NIST Cloud Deletion Use Case to securely Erase Data Objects in Cloud [9].

ORAM protects the access pattern from an observer. The seminal work on the topic is Goldreich and Ostrovsky [1], and since then, much work (e.g., [2, 3]) has focused on improving the efficiency of ORAM in space, time, and communication cost complexities. Moreover, considerable research work is conducted on storing data where cloud providers are untrusted [5, 6].

Researchers have developed individual oblivious data structures to accomplish specific tasks, such as priority queues, stacks and queues, and graph algorithms. Recently, Wang et al. [4] designed oblivious data structures (ODS) for maps, priority queues, stacks, and queues that were significantly more efficient than previous work and naive implementation of the data structures on top of ORAM. We use the techniques from Wang et al. for positional mapping [4] and extend (non-recursive) Path ORAM [6], to allow variable sized data items to be spread across multiple ORAM buckets.

While there are ontologies defining cloud security, cloud service life cycle, cloud service-level agreements and cloud compliance [12], there is very little work on defining ontology for oblivious cloud storage. We have developed an ontology, presented in section III, to describe the oblivious cloud storage using Oblivious Map Data Structure.

This project was funded by Office of Naval Research under grant N00014-15-1-2228

III. OBLIVIOUS MAP DATA STRUCTURE (ODS)

Our system uses an oblivious map data structure [21] as the backend to store files. Its primary features are a simple key-value architecture and strong privacy guarantee. As a key-value store, it represents a set of (key, value) items and supports inserting an item, searching for an item containing a given key, and deleting an item with a given key. In our context, a data item corresponds to a file, with the file-name as the keyword and the file contents as the value.

The map structure provides strong privacy guarantee of obliviousness. The network traffic to a server can reveal which raw blocks are being read and written to an attacker or the server itself. This access pattern may leak sensitive information about the underlying stored data, such as keyword search queries or encryption keys [18, 19, 20], even if the actual data is encrypted. Our data structure system will ensure that the server-level access pattern reveals nothing about the underlying data operations.

In addition to obliviousness, the ODS map system provides other beneficial security guarantees. In the cloud storage scenario, obliviousness will protect the client's privacy from any observer of network traffic or from the cloud server itself. However, *if the attacker compromises the client and obtains critical information such as the encryption keys used in the ODS,* all the sensitive information stored in the cloud will simply be revealed to the attacker. We call this scenario a catastrophic attack, and our system provides two security guarantees even against the catastrophic attack:

- Bounded History independence: In our system, the internal structure only reveals the information about recent operations, so old operations cannot be inferred by the structure.
- Secure deletion: In our system, the catastrophic attacker should not be able to guess information about data that has been deleted.

We significantly improved the efficiency of existing ODS maps by Wang et al. [4]. We report the system details and experiment results in a separate paper [21]. Here, we briefly overview the main components of the ODS map system.

A. System Components

Following the previous ODS system by Wang et al. [25], we implemented an ODS by constructing a tree-based data structure on a non-recursive ORAM. In order to improve efficiency and provide a better security guarantee, we designed a new tree-based data structure we call HIRB (history-independence randomized b-tree) and a new ORAM that admits variable size data blocks, called vORAM.

vORAM. The design of vORAM is based on the non-recursive version of Path ORAM [6], but we add more flexibility by allowing each ORAM bucket to contain as many variable-size blocks (or parts of blocks) as the bucket space allows. vORAM preserves obliviousness and maintains a small stash as long as the size of variable blocks can be bounded by a geometric probability distribution. To support secure deletion, we also store encryption keys within each bucket for its two children. The keys are re-generated on every access in a manner similar to other secure deletion frameworks [17].

The main observation is that the irregularity of block sizes can be smoothed over O(log n) buckets from the vORAM root to an vORAM leaf, where n is the maximum number of blocks that vORAM contains.

HIRB. We also developed a new data structure, called a HIRB tree (history independent, randomized B-tree). Conceptually, it is a fixed height B-tree such that when each item is inserted, the level in HIRB tree is determined by biased coin flips. The tree may split or merge depending on the situation, but *it never rotates*. Moreover, every operation visits at most 2H nodes, where H is the height of the tree, greatly saving on padding costs compared to previous ODS schemes.

IV. OBLIVICLOUDMANAGER APPLICATION

In this section, we describe the *ObliviCloudManager* application that we have developed to manage data on the cloud in an oblivious fashion such that the cloud provider cannot decipher it or determine its access or usage patterns.

A. Oblivious Cloud Ontology

In our ontology, every data object to be stored on the cloud is defined as a "document". The document can be of any format supported by the cloud provider.

All security policies, such as encryption policy, oblivious storage policy etc. are applied at the Collection level. Therefore all the documents in the collection automatically inherit the assigned policies of the Collection to which they belong. This saves time on selecting security policies every time a new document is added in the system. Based on cloud computing security guidelines from NIST [9, 10] and Cloud Security Alliance [16], we have developed oblivious cloud storage ontology which defines classes for Document, Collection, EncryptionPolicy, StoragePolicy, AuditPolicy, LoggingPolicy, KeyManagement, DeletionPolicy, User, ChunkingPolicy, ModificationPolicy, Permission, UserGroup, WhitelistedCloudProviders, WhitelistedVendors and others..

All the classes have data properties and object properties for detailed information. The data properties define the relation between individuals (instances) of OWL classes to literal values whereas object properties define relation between individuals (instances) of two OWL classes. For example hasDocuments property of Collection class defines a relation between a Collection instance and a Document insstance. Figure 1 shows some of the relations between instances of Oblivious Cloud Storage classes.

One of the benefits of incorporating an OWL ontology in our approach is to use an OWL reasoner to determine if there are any constraint violations in the Oblivious Cloud Storage Manager application data. Our implementation framework stores the ontology to relational database mapping tool and convert the database instances to RDF file and then reasons over it

This project was funded by Office of Naval Research under grant N00014-15-1-2228

to detect violations to the rules defined by cloud customer. Our choice to use a relational database backend was motivated by the expectation that most users would be more comfortable using a relational databases to store the policy model rather than a native RDF triple store.

As a simple example, consider a user who has a policy that documents in the HR-Document-Collection have key length of 64 whereas those in the UserManual-Collection have key length of 32. If she uploads a document in HR-Document-Collection with key length other than 64, reasoner will detect this constraint violation since the key length property is functional.

B. *Oblivious Cloud Manager*

The ObliviCloudManager is a web application that allows users to define and enforce policies on the collections and documents. Figure 2 illustrates the architecture of the application. Users access the application using a browser and it has role-based access control system for managing the User

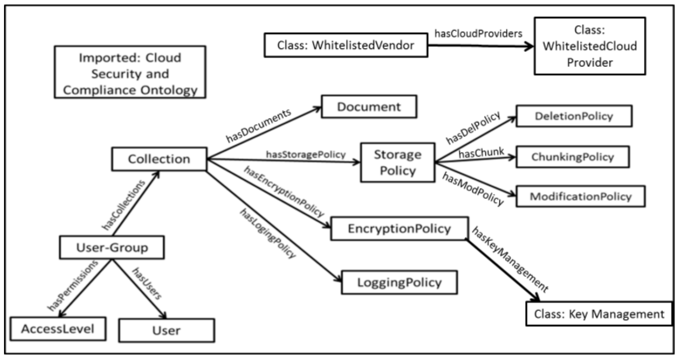


Figure 1. Ontology describing relation between the various classes in an Oblivious Cloud Storage

module. The application's backend uses a relational database structure which is mapped from the Ontology we defined in section IV A. Application resides on the ODS implementation described in section III.

All documents are stored using ODS and the root key of the vORAM is stored in HIRB and when a file is stored in vORAM the encrypted blocks of the file are then sent to cloud for storage. These blocks do not require any security while communicating to cloud, such as secure FTP connection, as it cannot be encrypted at all without the root key of vORAM which stays with the application. The application can be deployed on an in-house infrastructure of cloud customer or on a trusted private cloud. Oblivious cloud storage manager uses policies defined in the ontology and stores the documents in ODS only if all the constraints in ontology are met. This application acts as a client to oblivious storage system and manages how the data is stored on public cloud infrastructure. This scenario is similar to that considered by Stefanov and Shi in [8] and Williams et. al. in their PrivateFS system [7].

Every Document instance is stored as a part of Collection instance. A Document is stored in the Oblivious Data Structure based on the policies applied to the Collection it belongs to. A User-group will have different level of access to Collection such as read or write or admin. Users are assigned to a user-group, so all the permissions to that user-group are automatically given to the user. Access permissions can be either given to the user-group or to individual user. We can assign a single user to multiple user-groups, so that he can access different collections (of documents) as per her/his role.

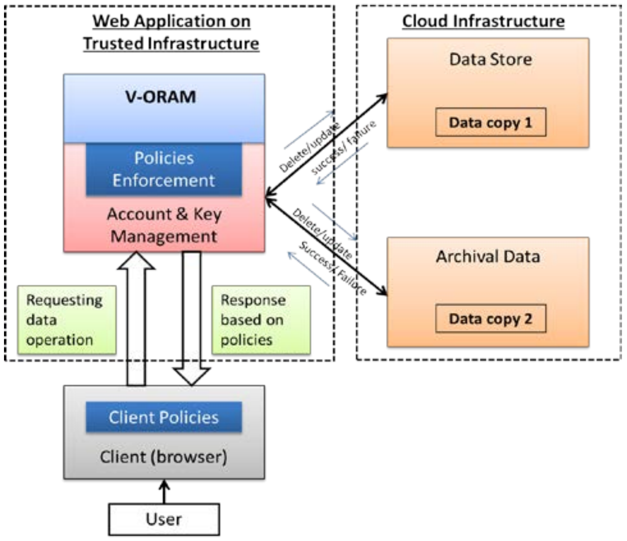
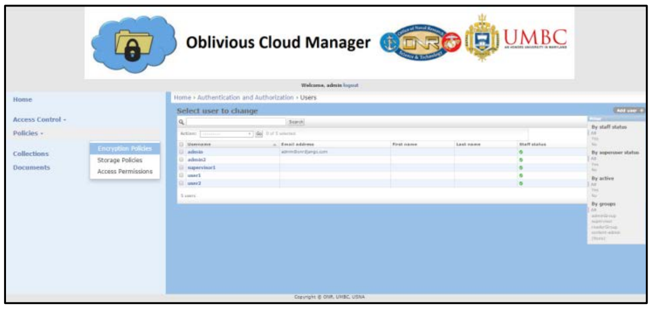


Figure 2. Oblivious Cloud Manager Architecture



Username	Email address	First name	Last name	Staff status
admin	admin@ontology.com			●
admin2				●
supervisor1				●
user1				●
user2				●

5 users

Copyright © ONR, UMBC, USNA

Figure 3. User Module

Figure 3 illustrates the application dashboard, which has four different modules shown at left panel: Access Control, Policies, Collections and Document modules. These modules are described below.

Access Control Module

The *Access Control Module* has two sub-modules: one for Users and another for Groups. Upon selecting Users, we can see a list of all users of the system and can add or edit users by accessing the appropriate user record.

Policy Module

The *Policy Module* is used to manage the consumer's policies on Encryption, Storage and Access Permissions. The Encryption policy module has all encryption policies defined for oblivious cloud store. Users can add/change encryption policy; it has all the fields defined by the ontology such as encryption algorithm, type, attribute/identity based encryption etc. It also

This project was funded by Office of Naval Research under grant N00014-15-1-2228

defines which type of key management is supported in this encryption policy. Upon clicking on add new key management policy, a form with details particular to encryption key details and storage pops up. Its fields include key length, key type, type of key storage, crypto period, etc. Similarly, the storage policy module has all storage policies defined for oblivious cloud store. The add/change storage policy form has fields defined in the ontology such as redundant storage, proof of storage protocol, proof of ownership, etc. It also defines which type of deletion policy, modification policy, and chunking policy is supported in this storage policy. Upon clicking on add new policy form is generated with respective details. Again these all fields in the form are based on ontology defined in section IV subsection A. The *Access Permission* module stores collection to user-group access permission mappings. It has a similar add new access permission form which accepts a Collection and User-group to access the collection and permission level..

Document Module

This module is used to manage the documents to be stored on the cloud obliviously. Document module shows list of documents that are stored in the system, collection to which the document belongs. On selecting a document from the list, it displays the details of that document such as collection, upload date, last modified date, user details, etc. When the details are entered by user, the save action initiates a process to store all the meta-data in the database and the actual document is stored in vORAM in HIRB. Figure 13 shows the access key when a new document is stored in oblivious cloud store. Note that, vORAM access key is shown for demo purpose only.

Collection Module

The Collection form accepts a collection name, date of creation, a description of the collection, root key to access the collection that is being stored as HIRB instance. The policies defined for the collection are applied to its documents while storing them in vORAM.

V. Conclusion and Ongoing Work

Many users need to ensure that their data stored in the cloud is private and secure and that patterns of its insertion, deletion and access will not reveal any information about it. We have created an ontology describing the oblivious cloud storage models and role based access controls to help manage such services. We have developed an algorithm to store cloud data using oblivious data structure and implemented an ObliviCloudManager application to support it. The application lets users to manage their cloud data more easily and effectively by specifying desired security constraints and policies associated with the oblivious data structures. Consumers of cloud services can use our system to define policies for storing data obliviously and manage storage on untrusted cloud platforms even if they are unfamiliar with the underlying technology and concepts of oblivious data structures.

Acknowledgment

This research was supported by the Office of Naval Research under grants N00014-15-1-2228 and N00014-16-WX-01489.

References

[1] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. J. ACM, 43(3):431–473, 1996.

[2] Ivan Damg°ard, Sigurd Meldgaard, and Jesper Buus Nielsen. Perfectly secure oblivious RAM without random oracles. In Yuval Ishai, editor, TCC 2011, volume 6597 of LNCS, pages 144–163. Springer, March 2011

[3] Michael T. Goodrich, Michael Mitzenmacher, Olga Ohrimenko, and Roberto Tamassia. Privacy-preserving group data access via stateless oblivious RAM simulation. In Yuval Rabani, editor, 23rd SODA, pages 157–167. ACM-SIAM, January 2012.

[4] Xiao Shaun Wang, Kartik Nayak, Chang Liu, T.-H. Hubert Chan, Elaine Shi, Emil Stefanov, and Yan Huang. Oblivious data structures. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, ACM CCS 14, pages 215–226. ACM Press, November 2014.

[5] K. Zhang, X. Zhou, Y. Chen, X. Wang, and Y. Ruan. Sedic: privacyaware data intensive computing on hybrid clouds. In CCS, 2011.

[6] E. Stefanov, M. van Dijk, A. Oprea, and A. Juels. Iris: A scalable cloud file system with efficient integrity checks. In ACSAC, 2012.

[7] P. Williams, R. Sion, and A. Tomescu. Privatefs: A parallel oblivious file system. In CCS, 2012.

[8] Emil Stefanov, Elaine Shi, ObliviStore: High Performance Oblivious Cloud Storage, In IEEE Symposium on security and Privacy, 2013.

[9] NIST Cloud Computing Security Reference Architecture, NIST Special Publication 500-299

[10] Elaine Barker, William Barker, William Burr, William Polk, and Miles Smid, Recommendation for Key Management – Part 1: General (Revision 3), 2012.

[11] NIST Cloud Computing Use Cases, Section 3 Cloud Management Use cases, sub-section 3.6. Erase Data Objects In a Cloud.

[12] A. Hendre, T. Finin, K.Joshi, Cloud Security and Compliance Ontology, July 2014, http://ebiq.org/r/361

[13] Karuna Pande Joshi et al., "Automating Cloud Services Lifecycle through Semantic Technologies", IEEE Transactions on Service Computing, January 2014.

[14] Natalya F. Noy and Deborah L. McGuinness, “Ontology Development 101: A Guide to Creating Your First Ontology”, Technical Report KSL-01-05, Stanford Knowledge Systems Laboratory, 2001.

[15] Julita Bermejo, “A Simplified Guide to Create an Ontology”, Technical Report R- 2007-004, Autonomous Systems Laboratory,, 2007.

[16] Cloud Security Alliance, “Security Guidance for Critical Areas of Focus in Cloud Computing V2.1”

[17] Joel Reardon, Hubert Ritzdorf, David Basin, and Srdjan Capkun. Secure data deletion from persistent media. In Ahmad-Reza Sadeghi, Virgil Gligor, and Moti Yung, eds, ACM CCS 13, pp.271–284. ACM Press, November 2013.

[18] J. L. Dautrich Jr and C. V. Ravishankar. Compromising privacy in precise query protocols. In Proceedings of the 16th International Conference on Extending Database Technology, pages 155–166. ACM, 2013.

[19] M. S. Islam, M. Kuzu, and M. Kantarcioglu. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In 19th Annual Network & Distributed System Security Symposium. The Internet Society, Feb. 2012.

[20] X. Zhuang, T. Zhang, and S. Pande. HIDE: an infrastructure for efficiently protecting information leakage on the address bus. In Proc. 11th Int. Conf. on Architectural Support for Programming Languages and Operating Systems , pp. 72–84, 2004.

[21] Daniel S. Roche, Adam J. Aviv, Seung Geol Choi. Practical Oblivious Map Data Structure with Secure Deletion and History Independence, IEEE Symposium on Security & Privacy (Oakland) 2016.

[22] Tim Mather, Subra Kumaraswamy, Shahed Latif, Cloud Security and Privacy: An Enterprise Perspective on Risks and Compliance , O'Reilly Media, Inc. , 2009

[23] D. Boneh, D. Mazi`eres, and R. A. Popa. Remote oblivious storage: Making oblivious RAM practical. Technical Report, CSAIL, MIT, 2011.

This project was funded by Office of Naval Research under grant N00014-15-1-2228