iDataGuard: Middleware Providing a Secure Network Drive Interface to Untrusted Internet Data Storage

Ravi Chandra Jammalamadaka†, Roberto Gamboni†, Sharad Mehrotra†, Kent E. Seamons‡, Nalini Venkatasubramanian
University of California, Irvine†, Brigham Young University‡,
{rjammala, gamboni, sharad, nalini}@ics.uci.edu seamons@cs.byu.edu

ABSTRACT

In this demonstration, we present the design and features of *iDataGuard*. iDataGuard is an interoperable security middleware that allows users to outsource their file systems to heterogeneous data storage providers available on the Internet. Examples of data storage providers include Amazon S3 service, Rapidshare.de and Nivarnix. In the iDataGuard architecture, data storage providers are untrusted. Therefore, iDataGuard preserves data confidentiality and integrity of outsourced information by using cryptographic techniques. iDataGuard effectively builds *a secure network drive* on top of any data storage provider on the Internet. We propose techniques that realize a *secure file system* over the heterogeneous data models offered by the diverse storage providers. iDataGuard significantly reduces the development effort required to build applications on top of the storage offered by the IDPs. Applications written to be compatible with iDataGuard, do not have to worry where the data is stored and how the security is enforced. iDataGuard automatically provides such functionality to application developers. To evaluate the practicality of iDataGuard, we implemented a version of the middleware layer to test its performance.

1. INTRODUCTION

Internet based data storage providers (IDP) have recently exploded on the market. Amazon S3 service, Yahoo Briefcase!, Megaupload.com are examples of such services. These services are based on a outsourcing model, where the clients outsource their data to IDPs, who provides data management tasks such as storage, access, backup, recovery, etc. The numerous benefits for the clients include: a) *Device Independence:* Clients can access their information from any machine connected to the Internet; and b) *Data Sharing:* The IDPs provide data sharing capabilities that allow users to share their data with any user on the Internet.

The primary limitation of such services is the requirement to *trust* the storage provider. The client's data *is stored in plaintext* and therefore is susceptible to the following attacks:

- **Outsider attacks:** There is always a possibility of Internet thieves/hackers breaking into the storage provider's system and stealing or corrupting the user's data.
- **Insider attacks:** Malicious employees of the storage provider can steal the data themselves and profit from it. There is no guarantee that the confidentiality and integrity of the user's data are preserved at the server side. Recent reports indicate that the majority of attacks are insider attacks [9, 8].

Despite these security concerns, IDPs are gaining popularity due to the convenience and usefulness of the data services they offer. In a related trend, there are many applications that are developed/being developed that leverage the storage infrastructures provided by the IDP services. Such applications are typically sold to organizations or individual users. Jungle disk [25] is an example of such an application which builds a network drive over the storage offered by Amazon S3. Herein lies the problem. Typically, organizations have service contracts with IDP services *of their choice* for their storage needs. Every IDP service provides its own interface for storing and fetching data within its infrastructure. There are no standards on interfaces for IDP services and therefore every service designs its own interface. For instance, in the Amazon S3 service, data is stored and accessed using the REST based protocol, and in Open X-Drive storage service the interface is based on the JSON-RPC protocol. The application developer now has the unenviable task of adapting his/her application to different IDP services, or otherwise risk significant reduction in clientele. This greatly complicates application development.

Imagine a trusted middleware that sits in between the application and the IDP service of choice. The middleware can run at a client machine or in a trusted proxy. The middleware provides a uniform interface for application development. The middleware can translate data related operations originating from the application to the equivalent operations at the IDP side. In other words, the burden of adapting to heterogeneity of interfaces is pushed from the application onto the middleware. Also, such a middleware can handle the data security requirements. Such a middleware greatly simplifies application development. This paper presents *iDataGuard*, a middleware that accomplishes the above goals.

EDBT'08, March 25–30, 2008, Nantes, France.

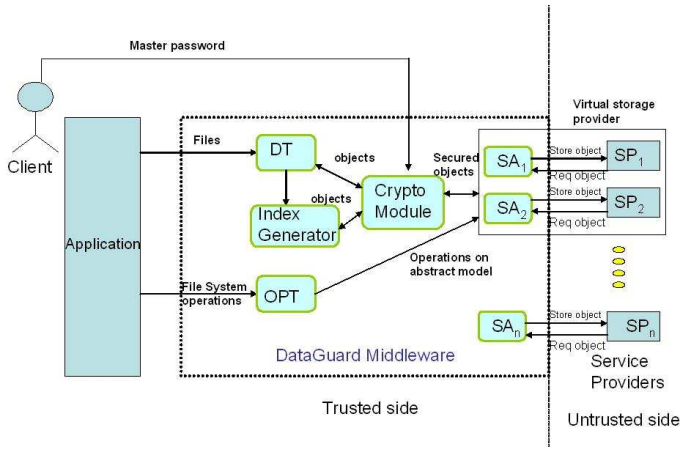

Figure 1: DataGuard Architecture

1.1 Architectural Overview

The desirable properties/goals of iDataGuard are the following:

- Allow users to outsource their file system to any Internet based data storage provider of their choice.
- Preserve security properties of user data such as data integrity and confidentiality.
- iDataGuard should be easy to use.

Major Entities and Threat Model: There are three main components in our architecture: a) Client machine; b) iDataGuard middleware; and c) Data storage providers. The Client machine is the end device from which the user is accessing the data. The client machine is entirely trusted. The iDataGuard middleware is trusted and runs inside the client machine. The middleware is in charge of providing data services to the user by fetching the required data from the storage providers. The storage providers provide data management services to the clients and are untrusted. We will assume an *honest-but-curious* behavioral model for the storage providers. That is, the storage providers are expected to provide the required services, but the employees that work for such providers could steal data and profit from it.

Overview: Figure 1 illustrates the overall architecture of iDataGuard. The outsourced file system from the application is first translated into an *abstract data model* by the middleware. Abstract data model is an object based model, generic enough to encompass the data models of a variety of IDPs. The abstract data model is explained in more detail in the next section. *Data translator (DT)* in fig 1 is in charge of that operation. The file system is mapped to a set of objects. The objects are then cryptographically secured by the *Crypto module*. The crypto module requires a secret provided by the client called the *masterpassword* to generate the cryptographic keys that are used to secure the objects. The objects are stored and fetched by the middleware by utilizing *service adapters*. Service adapters are IDP specific modules. For every service the middleware supports, a service adapters needs to be written. Service adapters utilize the interface (API) by the IDP to store and fetch objects. The *index generator* component generates the cryptographic index for all the textual files that are outsourced.

The application can issue all file system operations to iDataGuard. iDataGuard translates all file system operations into equivalent operations on the *abstract operation model*. Abstract operation model models a set of operations that are generic enough to encompass the operational model of a variety of IDPs. The abstract operation model is explained in more detail in the next section. The translation is done at the *operation translation (OPT)* component. The operations are first executed at the object level and their effects propagated to the IDPs via the service adapters.

iDataGuard also allows clients to specify a group of IDPs that can be treated as a single *virtual storage provider*. This flexibility does not complicate the design.

Who writes the service adapters? Service adapters need to be written by experts capable of understanding the interfaces/APIs provided by the service providers. Service adapters are fairly easy to write, requiring the implementation of only a few functions. Once the service adapters are developed, we envision that they will be freely distributed on the Internet. The client can procure such service adapters and install them in the middleware.

2. RESEARCH CONTRIBUTIONS

2.1 Adopting to Heterogeneity

iDataGuard allows users to specify which IDP they want to store their data. To provide such functionality, iDataGuard needs to take into account the heterogeneity of the data models that are offered by the IDPs. For instance, in Amazon S3 service, files are the basic units of data, while in Gmail.com, emails are the basic data units. One of the fundamental tenets of iDataGuard is make sure that *no changes are required at the server to support iDataGuard*. The servers are oblivious to the existence of iDataGuard.

To combat such heterogeneity, iDataGuard provides an *abstract service model*, that can then be further customized to individual IDPs. The abstract service model is designed to be generic enough to encompass all the IDPs that are currently functioning on the Internet. The abstract service model describes abstractly the data format and the functionality that IDPs need to provide to be compatible with iDataGuard. The abstract service model places *minimal requirements* on the functionality of the IDPs. Requiring IDPs to support more functionality could impede adoption of iDataGuard.

The abstract service model primarily consists of an *abstract data model* and an *abstract operation model*. The abstract data model models data as *objects*. Objects are atomic units of data. For instance, a pdf document or an image can be an object. Every object O has a unique id ($O.id$) and a set of attributes $O.A$ where $A = \{id, content, metadata\}$. $O.content$ represents the object's content and $O.metadata$ represents the ancillary information about the object. The metadata is a set of *attribute=value* pairs. The file system outsourced by the application is translated to a set of objects.

The *abstract operation model* consists of a set of operations that IDPs are expected to perform. It includes the *store_object(o)*, *fetch_object(o.id)* and *delete_object(o.id)* operations, where o is the object outsourced to the IDP. For a more detailed treatment of the abstract service model, please refer to [6].

2.2 Security Model

Since the IDPs are untrusted in our model, we propose a *security model* that allows iDataGuard to ensure data confidentiality and integrity of user's data by using cryptographic techniques. Our security model does not reveal: a) *Content of the files*; b) *Metadata of the file system*; and c) *Structure of the file system* to the server. It is obvious that the file content must be protected. We believe that mere encryption of the files is insufficient in itself. Both the metadata and the structure of the file system also contain a lot of information about user's data and therefore it makes sense to hide them as well. In this section we will briefly describe our security model.

In iDataGuard the file system outsourced is mapped to a set of objects. An object is created for every directory and file. The directory object contains a list of pointers to the children (files/other directories). When the objects are encrypted in a straightforward manner, the size of the encrypted directory object discloses the number of children it contains, hence revealing the structure partially. iDataGuard ensures that all encrypted directory objects are of equal size and thereby it prevents file structure leakage. This is achieved by padding small objects with irrelevant bits and breaking large objects into a set of equi-sized objects. Besides protecting file structure, iDataGuard also protects the integrity and confidentiality of objects in the following manner.

Confidentiality: For every data object(i.e., file/directory), iDataGuard generates an object encrytion key and encrypts the object with it. The object encryption key is generated using the master password and the object's name. The master password is the only secret the clients needs to remember to use iDataGuard. Generating a unique key for every object prevents cryptanalysis attacks.

Integrity: iDataGuard calculates integrity information for every object by using the HMAC primitive. To guarantee *freshness* of an object, iDataGuard keeps track of the object's version number. The HMAC is calculated using both the object's content and the version number. This is done so that the server does not cheat and retrieve an older version of the object when requested. Clearly, the version numbers cannot be stored at the server. To solve this problem, iDataGuard leverages the ability to store data with multiple storage providers. For instance, if a user configures iDataGuard with at least two different storage providers SP1 and SP2, then iDataGuard can store the version numbers of the objects belonging to SP1 with SP2 and vice versa. If we assume that SP1 and SP2 do not collude then the last update problem can be solved. This assumption of non-colluding servers has been made previously [3] and it is applicable to iDataGuard since the storage providers are in two different administrative domains to reduce the probability of an attack.

Our security model is designed to strike an appropriate balance between security and performance.

2.3 Search on Encrypted Data

iDataGuard supports all the operations supported by modern file systems such as creating a directory, reading a file, etc. iDataGuard also allows users to search for documents that contain a particular keyword. Such a task in the context of iDataGuard is very challenging, since the data is encrypted at the server. The obvious solution of fetching all the encrypted data from the server, decrypting it and executing the query locally is impractical as it puts tremendous performance strain on the system. We developed a *novel index based approach of executing such keyword based queries at the server.* The proposed index (CryptInd++) is carefully designed not to disclose any information to adversaries. Previous work [7, 4] on executing queries over encrypted data cannot be utilized in the context of iDataGuard, since the previous work assumes that the server is cooperative and runs a compliant protocol for enabling search. We cannot make such an assumption, since in the iDataGuard architecture, no changes are possible at the server. Our proposed index based approach does not require changes at the server. Any server that can store and fetch objects can support our index based approach for encrypted search. CryptInd++ also allows pattern based queries not supported by pervious approaches.

Overview of the index based approach: CryptInd++ is a double index. One index(Keyword index) maintains an inverted list of all unique keywords and set of all document ids that contain the keyword. Another index (q-gram index) maintains an inverted list of all unique q-grams and set of keywords that contain the q-gram. For instance, let us assume that only one document is indexed and it only contains one word "secure". Then, a keyword index entry $<$ "*secure*", "*document_id_outsourced*" $>$ is created. Note, all the values are encrypted and not present in plaintext as we have shown above. Then, the q-gram index with the following four index entries is created: $<$ "*sec*", $\{Secure\}>$, $<$ "*ecu*", $\{Secure\}>$, $<$ "*cur*", $\{Secure\}>$ and $<$ "*ure*", $\{Secure\}>$. The q-gram entries are also encrypted. When the client wants to search for all documents that contain a word that matches the pattern "*cur*", the q-gram entry $<$ "*cur*", $\{Secure\}$ is fetched first and then subsequently $<$ "*secure*", "*document_id_outsourced*" $>$. The document id will be the result of the query. The above example is a very simple. If a pattern contains more than one q-gram, all the q-gram entries for the q-grams present in the pattern are fetched from the IDP. The intersection of the keywords present in the q-gram entries is calculated. These are keywords that can potentially match the pattern. These keywords are then checked if they match the pattern locally at the trusted client side. Let K_p be the set of keywords that match the pattern. For all the keywords in K_p, the document ids that contain the keywords are retrieved from the server and subsequently the required documents. All the above steps are explained in greater detail later in [6].

3. DEMONSTRATION SCENARIO

Our demonstration shows how a client application written to access the iDataGuard middleware can utilize the storage provided by the Amazon S3 service and Gmail.com. Although Gmail.com provides an email service on top of the storage, we ignore that service in the interest of the storage space. The application is incognizant of where the data is stored and it is up to the client/user to dictate such requirements.

We developed a File Backup application that allows clients to backup their files and directories to any IDP that is compatible with iDataGuard. File Backup provides full file system functionality (e.g., creating a directory, deleting a file) to the user. The demonstration will include a client machine, such as a standard laptop, connecting to Internet based data

storage providers. The features of the File Backup client application will demonstrated for both the client and server side representations of the data, since data is secured before it leaves the client perimeter. Fig 2 illustrates a snapshot of the File Backup application that utilizes the iDataGuard middleware.

4. RELATED WORK

Network file systems [18, 19, 20] allow users to outsource their information to a remote server. An authorized client can then mount the file system stored at the server. Typically in these systems, the server is trusted and is in charge of authentication of the users and enforcing access control on data. This is not the case in iDataGuard.

Cryptographic file systems [2, 15, 13, 14] on the other hand are very related to our work. Cryptographic file systems do not trust the end storage and all the cryptographic operations are done at the trusted/client side. Cryptographic file systems such as Sirius [13] and Plutus [14] also allow sharing of files between users, where access to files is provided via key distribution. iDataGuard currently does not deal with sharing, although it is one of our future goals. We differ from the cryptographic file systems in the following manner: a) cryptographic file systems do not adopt to the heterogeneity of data models of the server side. Typically, they assume a file system based model at the server. iDataGuard on the other hand can easily adapt to the heterogenous data models at the server.

DAS [11, 12] architectures allow clients to outsource structured databases to a service provider. The service provider now provides data management tasks to the client. The work on DAS architecures mainly concentrated on executing SQL queries over encrypted data. The clients of DAS architectures are mainly organizations that require database support. Both the DAS architectures and iDataGuard can be thought of as instantiations of the outsourced database model (ODB). The key differences are: a) The data outsourced in DAS is highly structured. In iDataGuard, the data oursourced is semi-structured. b) DAS architectures did not deal with mobility issues, which is one of the primary goals of iDataGuard.

Distributed file systems like Oceanstore [10] provide a storage infrastructure for the users to store data on the network rather than at a centralized server. In ocenstore, the files are treated as objects and are replicated across multiple locations. The goal is to ensure availability, scalability and fault tolerance. iDataGuard should not be treated as a distributed file system, since middleware treats the storage providers as a single logical entity. This does not imply that the service providers do not implement a distributed storage infrastructure. On the other hand, iDataGuard does allow users to mount different file systems with multiple storage providers.

In DataVault [21], the authors proposed a client-server architecture which allows users to outsource their file systems to an untrusted server. The server then provides data services on top of outsourced data. iDataGuard is not a client-server architecture, it is a middleware that is trying to utilize the storage space provided by untrusted servers on the Internet. In DataVault, the authors were able to design a server architecture from scratch that suits their data storage requirements. iDataGuard middleware on the other hand has to work/adapt to the current data storage infrastructures of the IDPs.

Jungle disk software[25] layers a security mechanism over the Amazon S3 storage service. Unlike iDataGuard, Jungle Disk can only function with the Amazon S3 service. Jungle Disk also provides a file system like interface to the user and preserves data confidentiality of the user by encrypting the data stored remotely. The user can provide a password as the key to encrypt the data. To the best of our knowledge, Jungle Disk does not verify the integrity of the data.

5. CONCLUSIONS

In this paper we presented iDataGuard, a novel interoperable security middleware that reduces the burden of developing applications that leverage the storage infrastructures provided by internet based data storage providers. To the best of our knowledge, this is the first research work that addresses security and heterogeneity issues with regard to IDPs at the middleware level. iDataGuard is a useful tool for both application developers and clients. iDataGuard ensures the confidentiality and integrity of the client's data. iDataGuard utilizes a novel index based approach to allow keyword based search on encrypted data. Our experiments indicate that performance degradation due to the middleware is negligible when compared to the network costs, thereby making the middleware approach practically feasible.

Our main goal is to release the iDataGuard middleware as open source software that will allow experts or webmasters themselves to write service adapters compatible with iDataGuard. We have currently developed an iDataGuard prototype that can be downloaded at `http://DataGaurd.ics.uci.edu`.

6. REFERENCES

[1] R.C.Jammalamadaka, T. van der Horst, S. Mehrotra, K.Seamons, and N. Venkatasuramanian. Delegate: A Proxy Based Architecture for Secure Website Access from an Untrusted Machine. Proceedings of 22nd Annual Computer Security Applications Conference (ACSAC), 2006.

[2] M.Blaze. A cryptographic file system for UNIX. Proceedings of the 1st ACM conference on Computer and Communications Security, 1993.

[3] G. Aggarwal, M. Bawa, P. Ganesan, H. Garcia-Molina, K. Kenthapadi, R. Motwani, U. Srivastava, D.Thomas, and Y.Xu. Two Can Keep a Secret: A Distributed Architecture for Secure Database Services. 2nd Biennial Conference on Innovative Data Systems Research (CIDR), 2005.

[4] E.j.Goh. Secure Indexes. Stanford University Technical Report. 2003

[5] RSA Laboraties. PKCS #5 V2.1: Password Based Cryptography Standard. `ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-5v2/pkcs5v2\_1.pdf`, 1999.

[6] R.C.Jammalamadaka, R. Gamboni, S. Mehrotra, K. Seamons, and N. Venkatasubramanian. iDataGuard: An Interoperable Security Middleware for Untrusted Internet Data Storage. University of California, Irvine, Technical Report, 2007.

[7] D. Song, D.Wagner, and A. Perrig. Practical Techniques for Searches on Encrypted Data. Proceedings of IEEE Symposium on Research in Security and Privacy, 2000.

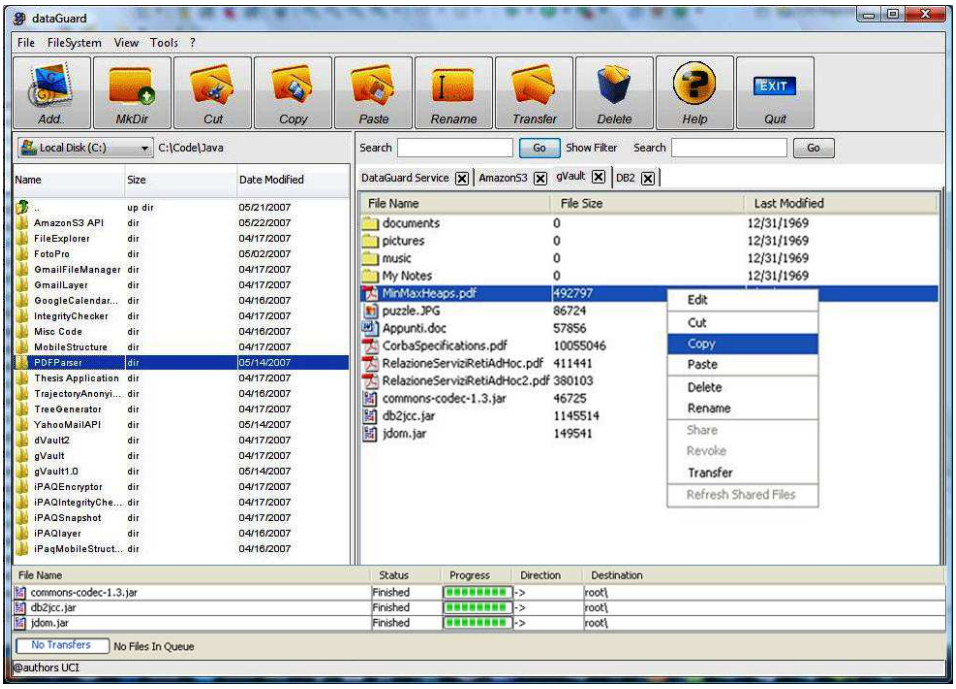

Figure 2: Snapshot of the File Backup application. The interface is similar to the one provided by modern operating systems. iDataGuard is implemented in Java.

[8] A. Briney. The 2001 Information Security Industry Survey 2001, 2002. http://www.infosecuritymag.com/archives2001.shtml

[9] G. Dhillon and S. Moores. Computer Crimes: Theorizing about the Enemy Within. Computers & Security 20 (8):715-723, 2001.

[10] S.Rhea, P.Easton, D.Geels, H.Weatherspoon., B.Zhao, and J.Kubiatowicz. Pond: The oceanstore prototype. In the proceedings of the Usenix File and Storage Technologies Conference (FAST) 2003.

[11] H.Hacigumus, B.Iyer, C. Li, and S. Mehrotra. Executing SQL over Encrypted Data in the Database-Service-Provider Model. ACM SIGMOD Conference on Management of Data, Jun, 2002.

[12] E.Damiani, S. De Capitani Vimercati, S.Jajodia, S. Paraboschi, and P.Samarati. Balancing confidentiality and efficiency in untrusted relational DBMSs. Proceedings of the 10th ACM Conference on Computer and Communications Security, 2003.

[13] E.Goh, H.Shacham, N.Modadugu, and D.Boneh. SiRiUS: Securing remote untrusted storage. Proceedings of Network and Distributed Systems Security (NDSS) Symposium, 2003.

[14] M. Kallahalla, E. Riedel, R. Swaminathan, Q. Wang, and K. Fu. Plutus: Scalable secure file sharing on untrusted storage. Proceedings of 2nd USENIX Conference on File and Storage Technologies (FAST), 2003.

[15] E.Zadok, I.Badulescu, and A.Shender. Cryptfs: A Stackable vnode level encryption file system. Technical Report CUCS-021-98, Columbia University, 1998.

[16] G.Miklau and D.Suciu, Controlling Access to Published Data Using Cryptography. Proceedings of Very Large Data Bases (VLDB) Conference, 2003.

[17] E.Bertino, B.Carminati, E.Ferrari, B.Thuraisingham and A.Gupta. Selective and authentic third party distribution of XML documents. IEEE Transactions on Knowledge and Data Engineering, Volume 16 , Issue 10, October 2004.

[18] S. Shepler, B.Callaghan, D.Robinson, R.Thurlow, C.Beame, M. Eisler, and D. Noveck. NFS version 4 protocol. RFC 3530, April 2003.

[19] J.Howard. An Overview of the Andrew File System. Proceedings of ACM Symposium on Parallel Algorithms and Architectures (SPAA), 2002.

[20] D. Mazires. Self-certifying file system. Phd Thesis, Massachusetts Institute of Technology, 2000.

[21] R.Jammalamdaka,S.Mehrotra, K.Seamons and N.Venkatasubramanian. Providing Data Sharing as a Service. University Of California, Irvine, Technical Report, 2007.

[22] http://www.aws.amazon.com/s3

[23] http://www.apple.com/dotmac/

[24] R.C. Jammalamadaka, R. Gamboni, S. Mehrotra, K. Seamons and N. Venkatasubramanian. gVault: A Gmail Based Cryptographic Network File System. Proceedings of 21st Annual IFIP WG 11.3 Working Conference on Data and Applications, 2007.

[25] http://jungledisk.com