Middleware Support for Protecting Personal Data from Web Based Data Services

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

ABSTRACT

Web based data services are very popular with the average computer user. Examples of such services include Gmail.com, Yahoo Photos, Yahoo Briefcase and Amazon S3 Service. In such services, the user outsources personal data to service providers who provides data management services on outsourced data. Such services have many advantages which include: a) *Mobile access:* The data can be accessed from any computer connected to the internet; b) *Availability:* the data is available 24/7; c) *Good service:* Typically such services employ experts who provide a quality service. However, such a model does raise some fundamental questions concerning data privacy and security. The data is stored in plaintext at the service provider and is vulnerable to data theft from disgruntled employees and internet thieves.

This paper describes our research in designing middleware architectures that secure personal data using cryptographic techniques before it is outsourced to the service provider. Care is taken such that service provider can continue to provide data services on secured data. We describe the challenges in designing and implementing such middleware architectures, a summary of our past work and the future directions that we intend to take with regard to the project.

1. INTRODUCTION

Recently, there has been an explosion in the number of web based data storage providers (WDP) that are emerging. Examples of such services include: Rapidshare.de, Youtube.com, Megaupload.com, Yahoo Briefcase!, Amazon S3 service, etc. The clients outsource their data to WDPs, who provides data management tasks such as storage, access, backup, recovery, etc. WDPs offer numerous benefits to users, which include: a) *Device Independence:* Clients can access their information from any machine connected to the Internet; and b) *Data Sharing:* The WDPs provide data sharing capabilities that allow users to share their data with any user on the Internet.

Currently, users employ a variety of ways to achieve mobility when it comes to personal data. The range of solutions include but are not limited to: i) Carrying their data in secondary storage devices such as USB drives, CD/DVDs, etc. This is a largely inconvenient solution pushing the burden of data management to the user. Also, the solution is inherently insecure, as most users store their data in plaintext. Such devices can be easily lost or stolen; ii) Maintaining public servers such as web servers, FTP servers, etc. The drawbacks of this solution are twofold: a) Administering such a service is burdensome and requires sound technical knowledge; and b) Many users are not in a position to run such a service due to ISP restrictions. Likewise, to share data, users employ solutions like sending email, etc., which suffer from similar drawbacks listed above.

By comparison, services offered by the WDPs do not suffer from the above drawbacks and have the following advantages: a) *Availability:* Data is available 24/7 from any computer connected to the Internet; b) *Low cost:* Typically, the services are free. The business model is based on advertisements, emphasizing the fact that storage has become very cheap; c) *Good Service:* The storage providers typically employ experts, thereby providing very high quality service. All of the above advantages make WDPs an attractive prospect for data storage.

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 [7].

There also have been instances of WDPs collaborating with repressive regimes and providing them personal data that belongs to users [18]. Fear of prosecution is in itself a valid reason not to trust WDPs with our personal data. As such,

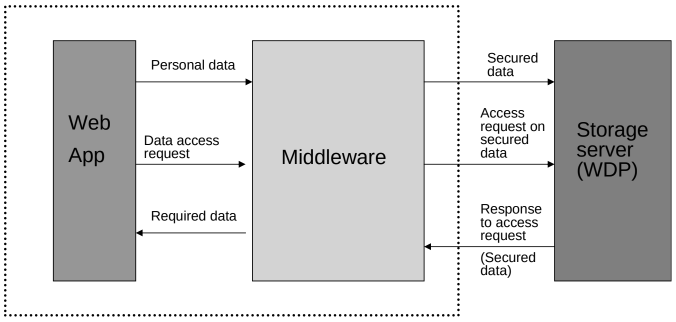


Figure 1: High level interactions of various components

use of WDPs for storing potentially sensitive information has been heavily limited. Despite these security concerns, WDPs are gaining popularity due to the convenience and usefulness of the data services they offer.

There are two major components involved in a WDP service, they are a :) *web application*, which utilizes the data storage provided by the service and runs at the client side; and b :) *storage server*, which runs at the service provider, where the data is actually stored. *Imagine a transparent middleware that acts as a middleman between the web application and the storage server.* The middleware secures the data before it leaves the client machine. The middleware ensures that the service provider can continue to provide the services on secured data. When the data is fetched from the service provider, the data is transformed back to its original form and presented to the user. The user now does not have to worry about security breaches with regard to its data. The goal of our research is to build such middleware architectures. In this paper, we will describe the challenges in building such architectures, our preliminary results and the future directions that we intend to take. Fig 1 illustrates the relevant high level components and their interactions.

1.1 Two service models

There are two different types of service models with regard to WDPs. They are: a) *Data storage only (DS)* service; and b) Application and data storage (ADS) service.

DS service model: In DS service model, the WDP only provides data storage and backup services. The web application or the application logic is not provided by the WDP. Examples of DS service model include Amazon S3 service and Gmail.com. Gmail.com also provides a web based email application, but such an application can be ignored in the interest of storage space. The WDP provides interfaces (API) that application programmers can utilize for their storage needs. As you can imagine, a variety of applications can be built on top of such storage. The biggest problem with the current setup is the heterogeneity in the storage models provided by the different WDPs. For instance, Amazon S3 service provides an API based on a proprietary object based data abstraction. In Gmail.com, data abstraction is in the form of emails. To achieve portability across many such WDPs, the web application programmer needs to worry about all the different data abstractions. This could be cumbersome for the application programmer. We designed and implemented the DataGuard middleware that besides enforcing data security, also takes care of the heterogeneity of the storage models provided by the WDPs. The DataGuard middleware is explained in more detail in section 2.

ADS service model: In the ADS service model, the WDP provides both the storage and the web application that utilizes the storage. Yahoo Photos is an example of a service that conforms to the ADS model. Most ADS services can also be considered as DS services by ignoring the web application. For instance, Gmail.com is an example of ADS service model, that can very easily considered as a DS service.

To develop a security middleware for ADS services, we need to address completely different set of challenges. The data transfer takes place via HTTP requests between the web application and the storage server. The middleware should analyze the HTTP requests, extract the sensitive information, secure the information using cryptographic techniques and then pass it on to the storage servers. Unlike DS services, the ADS services typically do not provide an API for data storage. Consider Yahoo Photos service, only a few HTTP requests actually carry the user's pictures. These HTTP requests need to be secured before being sent to the web server. Other HTTP requests can be ignored.

It will be very difficult to develop automatic techniques that can determine when HTTP requests carry personal data. The HTTP requests are not semantically rich enough to help the middleware in this regard. We envision a middleware that takes the help of domain knowledgeable experts. The experts teach the middleware where look for sensitive data using appropriate interfaces. Section 3 provides more information about the envisioned middleware which is a part of our future work.

2. DATAGUARD MIDDLEWARE

In this section, we will briefly summarize the goals and design of the DataGuard middleware, that allows the users to outsource their data to untrusted WDPs. Our goal is to develop a middleware that a client can run on their local machines, which can interact with the WDPs of their choice and yet manage the client's data securely. DataGuard address the problem at the file level, i.e., the users outsource their local file system to the WDPs. DataGuard effectively builds a network drive on top of data storage provided by an WDP.

There are three primary reasons that motivates our work on designing such a middleware: a) **Popularity:** Network drives are very popular because they allow users remote access to their data. They effectively provide a virtual disk that users can carry around seamlessly without much effort. This is precisely the reason why there are many commercial WDPs offering a *network drive like* service on the Internet [15, 16]; b) **Security:** Data should be secured before it is outsourced to an untrusted server. c) **General applicability:** A wide variety of applications can be supported by a *file storage* like service. For instance, consider the following sample applications that can be supported: a) an *autofill application*, which remembers and fills out passwords from any machine connected to the Internet. b) a *bookmark manager* which provides remote access to personal bookmarks.

2.1 Research Contributions

Our research in designing DataGuard has led to the following contributions. Here, we will describe them briefly, for more information please refer to the full version of the DataGuard paper [5].

Heterogeneity: DataGuard allows users to specify which WDP they want to store their data. To provide such functionality, DataGuard needs to take into account the heterogeneity of the data models that are offered by the WDPs. 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 DataGuard is make sure that *no changes are required at the server to support DataGuard*. The servers are oblivious to the existence of DataGuard. To combat such heterogeneity, DataGuard provides a novel general model of a file/data, that can then be further customized to individual WDPs. We will call this model as the *generic data model* (GDM). We will propose techniques to map the generic database model to server side data representation.

Security Model: Since the WDPs are untrusted in our model, we propose a *security model* that will allows DataGuard to ensure data confidentiality and integrity of user's data by using cryptographic techniques. The cryptographic keys are generated by using a secret called the *masterpassword* known only to the user. Masterpassword is the only secret the user is expected to remember which makes DataGuard easy to use.

Cryptographic Index: DataGuard supports all the operations supported by modern file systems such as creating a directory, reading a file, etc. DataGuard also allows users to search for documents that contain a particular keyword, a challenging task since 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 develop a novel index based approach of executing such keyword based queries at the server. The proposed index is carefully designed not to disclose any information to adversaries. Previous work [6, 3] on executing queries over encrypted data cannot be utilized in the context of DataGuard, 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 DataGuard architecture, no changes are possible at the server.

2.2 Future Work/Open Problems

In this section we will describe some of the open problems in DataGuard. The following problems will be the main focus for our future work in DataGuard.

Data Sharing: Currently DataGuard allows users to access their data remotely. WDPs also offer data sharing functionality. DataGuard needs to be extended to leverage such functionality and provide users with secure mechanisms to share data with other users. Data sharing needs to be done in a fashion where the server does not learn any of user's data.

Accessing Information from Untrusted Machines: DataGuard currently assumes that all end devices the user access his/her data are trusted. While in most cases this is true, in some cases it isn't. For instance, consider Alice who is traveling without a laptop. She needs to access her data from a publicly accessible machines such the ones that are avaialble in cybercafe or a public machine. Such public machines can harbor macicious entities which could steal Alice's masterpassword. A simple keystroke logger will accomplish the job. Clearly, this is undesirable. We need techniques to to access personal data from untrusted machines. In other words, DataGuard needs to be extended to allow access to data from untrusted machines. In [19] the authors propose a proxy based solution to access websites securely from untrusted machines. We envision a similar solution could solve the above problems.

3. ADS SERVICE MODEL

Recall that in ADS service models both the web application and storage servers are provided by the WDPs. The interaction between the web application and the storage servers is via HTTP requests. Typically, data is sent via the HTTP POST requests. A security middleware that wants to protect user's data, should intercept the HTTP requests, identify the sensitive information from the requests and then secure the information. The secured information should be placed back in the HTTP requests and then forwarded to the server. The middleware should also: a) *be transparent to the user*, i.e., the user should be cognizant of the middleware existence. In other words, the web experience of the user should not be changed; and b) *not significantly reduce the performance of the web application.* This is important, as users tend to disable security measures when confronted with performance delays.

We will explain the different challenges in building a middleware of this kind.

Identifying sensitive information: All the HTTP requests sent to the server are not sensitive. Only a very small fraction of the requests are sensitive. For instance, in Yahoo Photos, only the HTTP POST requests that forward the personal pictures should be considered sensitive. The identification process can be made automatic, where the middleware utilizes some heuristics to determine if the HTTP request is sensitive or not. But such a model, can be unreliable if it fails to secure some sensitive information. We are of the opinion, that experts with domain knowledge can train the middleware in identifying the sensitive information that is sent back and forth in a web service. The middleware needs to be trained individually for each web service/website. The experts generate *rules* specific to a website which can be plugged into the middleware. Once the experts, generate the rules, the rules can be distributed freely on the internet. Interested users can then download the rules and install them in the middleware. We are currently developing a rule specification language that will allow experts to easily create the rules. Experts could be anybody with enough technical expertise to analyze the HTTP requests, which includes the webmasters themselves.

Securing the sensitive information: Once the sensitive HTTP requests are identified, the requests needs to be secured using cryptographic techniques. Middleware should ensure that data confidentiality and integrity of the sensitive information is preserved. The sensitive information can be encrypted with cryptographic keys that are derived from a masterpassword. The masterpassword is the only secret that the user needs to be provide the middleware at the beginning of a session. The middleware also needs to compute data signatures and store them along with the secured data to ensure data integrity. Also, the encrypted information needs to be tagged with *metadata* that will allow the middleware to identity the encrypted data when it is shipped back from the server. The middleware then can decrypt the data and show the plaintext from of the data to the user. We are currently investigating efficient methods to enforce data confidentiality and integrity constraints at the HTTP request level.

Data sharing: Most of the ADS services offer data sharing functionality. For instance, Yahoo Photos allows users to share pictures with others. The bulk of the work in data sharing architectures comes from authentication and data distribution. Current WDPs already perform such operations. We want to retain the semantics of data sharing at the WDPs and yet share data in a fashion that will not allow the server to learn sensitive information. For example, we would like pictures to be shared among users in Yahoo Photos, without the website learning the actual pictures. We are currently investigation techniques that will allow secure sharing to take place via ADS services.

4. RELATED WORK

We will first discuss DataGuard related work. Network file systems [12, 13, 14] 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 the middleware architectures that we are investigating.

Cryptographic file systems [1, 10, 11] 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 [10] and Plutus [11] also allow sharing of files between users, where access to files is provided via key distribution. DataGuard 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. DataGuard on the other hand can easily adapt to the heterogenous data models at the server.

DAS [8, 9] 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 DataGuard 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 DataGuard, the data outsourced is semi-structured. b) DAS architectures did not deal with mobility issues, which is one of the primary goals of DataGuard.

Distributed file systems like oceanstore [21] 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. DataGuard 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, DataGuard does allow users to mount different file systems with multiple storage providers.

In DataVault [20], 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. DataGuard 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. DataGuard middleware on the other hand has to work/adapt to the current data storage infrastructures of the WDPs.

Jungle disk software[17] layers a security mechanism over the Amazon S3 storage service. Unlike DataGuard, 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.

To the best of our knowledge there is no similar academic work related to the ADS service middleware that we are envisioning.

5. CONCLUSIONS

In this paper we motivated the requirement to secure the personal data before outsourcing it to web based data storage providers. We advocated a middleware based solution to address the security concerns. We introduced the DataGuard middleware that allows users to outsource their data to heterogenous storage providers on the Internet. We also presented our envisioned ADS service middleware and the challenges inherit in it. We stated some of the open problems that we have discovered and the future directions we intend to take. A preliminary version of the DataGuard middleware that can be downloaded at http://DataGuard.ics.uci.edu. We encourage users to try it and provide us with their valuable feedback.

6. ACKNOWLEDGEMENTS

We acknowledge the work done by Roberto Gamboni in implementing parts of the DataGuard middleware. This research was supported by funding from the National Science Foundation under grant no. CCR-0325951 and IIS-0220069, the prime cooperative agreement no. IIS- 0331707, and The Regents of the University of California.

7. REFERENCES

[1] M.Blaze. A cryptographic file system for UNIX. Proceedings of the 1st ACM conference on Computer and communications security.

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

[3] Eu jin Goh. Secure Indexes. In submission

[4] RSA Laboraties. PKCS #5 V2.1: Password Based Cryptography Standard. ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-5v2/pkcs5v2_1.pdf

[5] Ravi Chandra Jammalamadaka, Roberto Gamboni, Sharad Mehrotra, Kent Seamons, Nalini Venkatasubramanian. DataGuard: A Middleware Layer Providing Seamless Mobile Access to Personal Data via Untrusted Servers. Techincal Report

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

[7] Dhillon, Gurpreet, and Steve Moores. 2001. Computer crimes: theorizing about the enemy within. Computers & Security 20 (8):715-723.

[8] Hakan Hacigumus, Bala Iyer, Chen Li, and Sharad Mehrotra. Executing SQL over Encrypted Data in the Database-Service-Provider Model. *2002 ACM SIGMOD Conference on Management of Data, Jun, 2002.*

[9] E.Damiani, S. De Capitani Vimercati, S.Jajodia, S. Paraboschi, P.Samarati. Balancing confidentiality and efficiency in untrusted relational DBMSs. Proceedings of the 10th ACM conference on Computer and communications security.

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

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

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

[13] J.Howard. An overview of the andrew file system. In proceedings of ACM symposium on parallel algorithms and architectures. SPAA, 2002.

[14] David Mazières. Self-certifying file system. Phd Thesis. 2000

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

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

[17] http://www.JungleDisk.com

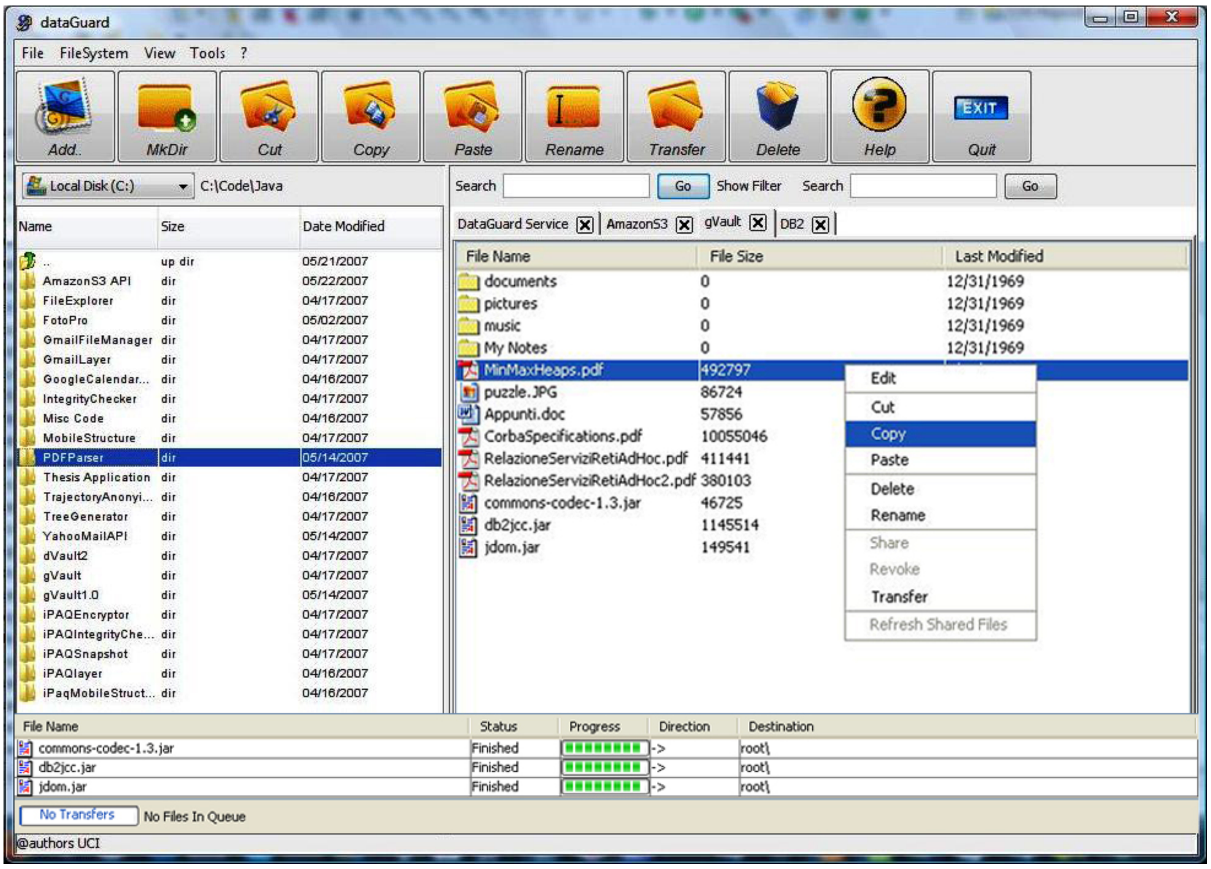
[18] Man Jailed after Yahoo Handed Draft Email to China. http://www.ctv.ca/servlet/ArticleNews/story/CTVNews/20060419/

[19] Ravi Chandra Jammalamadaka; Timothy van der Horst; Sharad Mehrotra; Kent Seamons; NaliniVenkatasuramanian. Delegate: A Proxy Based Architecture for Secure Website Access from an Untrusted Machine. 22nd Annual Computer Security Applications Conference (ACSAC), Maimi, FL, December, 2006.

[20] Ravi Chandra Jammalamadaka, Sharad Mehrotra, Kent Seamons, Nalini Venkatasubramanian. DataVault: An Architecture Providing Secure Mobile Access and Data Sharing on the Web. UCI Technical Report.

[21] 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.

8. APPENDIX



C:\Code\Java

Name	Size	Date Modified
..	up dir	05/21/2007
AmazonS3 API	dir	05/22/2007
FileExplorer	dir	04/17/2007
FotoPro	dir	05/02/2007
GmailFileManager	dir	04/17/2007
GmailLayer	dir	04/17/2007
GoogleCalendar...	dir	04/16/2007
IntegrityChecker	dir	04/17/2007
Misc Code	dir	04/16/2007
MobileStructure	dir	04/17/2007
PDFParser	dir	05/14/2007
Thesis Application	dir	04/17/2007
TrajectoryAnonyi...	dir	04/16/2007
TreeGenerator	dir	04/17/2007
YahooMailAPI	dir	05/14/2007
dVault2	dir	04/17/2007
gVault	dir	04/17/2007
gVault1.0	dir	05/14/2007
iPAQEncryptor	dir	04/17/2007
iPAQIntegrityChe...	dir	04/17/2007
iPAQSnapshot	dir	04/17/2007
iPAQlayer	dir	04/16/2007
iPaqMobileStruct...	dir	04/16/2007

File Name	File Size	Last Modified
documents	0	12/31/1969
pictures	0	12/31/1969
music	0	12/31/1969
My Notes	0	12/31/1969
MinMaxHeaps.pdf	492797	
puzzle.JPG	86724	
Appunti.doc	57856	
CorbaSpecifications.pdf	10055046	
RelazioneServiziRetiAdHoc.pdf	411441	
RelazioneServiziRetiAdHoc2.pdf	380103	
commons-codec-1.3.jar	46725	
db2jcc.jar	1145514	
jdom.jar	149541	

File Name	Status	Progress	Direction	Destination
commons-codec-1.3.jar	Finished	🟩🟩🟩🟩🟩🟩🟩🟩	->	root\
db2jcc.jar	Finished	🟩🟩🟩🟩🟩🟩🟩🟩	->	root\
jdom.jar	Finished	🟩🟩🟩🟩🟩🟩🟩🟩	->	root\

Figure 2: Snapshot of the DataGuard application/middleware. The interface is similar to the one provided by the modern operating systems. DataGuard is implemented in Java.