2013 IEEE 16th International Conference on Computational Science and Engineering

Secure and Controlled Sharing of Data in Distributed Computing

Danan Thilakanathan, Rafael Calvo
The University of Sydney
NSW, Australia
firstname.lastname@sydney.edu.au

Shiping Chen, Surya Nepal
CSIRO ICT Centre
Marsfield, NSW
firstname.lastname@csiro.au

***Abstract*—In today's world, there is a strong need for sharing big data for collaboration. However, privacy and security remains a barrier especially when dealing with massive amounts of data in the Cloud. In particular, there is a growing need for the data owner to gain better access control over their data. The data owner should be able to specify how their data should be viewed, copied and modified. A dishonest authorised user may illegally redistribute the data to other friends and/or colleagues who don't have the relevant access permissions. In this paper, we address these issues by introducing the idea of *SafeShare* which controls how data can be accessed and used. SafeShare encapsulate data in self-controlling objects (SCO's) and additionally monitors operations of the user. We demonstrate that our SafeShare system will prevent the user from carrying out any illegal operations including illegal redistribution of data. In addition, we add another level of security on the SCO's by adding a log file and hash for better auditing control.**

***Keywords*-Distributed Computing; Cloud; Big Data Sharing; Access Control**

I. Introduction

There is a growing trend among people today to share and collaborate large amounts of data with each other [1]. Businesses and organisations benefit through greater productivity and efficiency when big data is shared or exchanged with business partners around the world using Cloud technology [2][3]. Millions of social users are also benefitting through greater enjoyment and an enrichment of life when sharing personal photos, videos and stories with friends and family using Facebook and Twitter [4]. With the growing improvements and accessibility of technology, it is expected that the demand for sharing massive amounts of data using a variety of new technologies will become prominent in future generations.

Although the future of big data sharing and collaboration in distributed systems seems promising, privacy and security issues continue to be a barrier to its progress [5][6], especially when dealing with big data stored in the Cloud [7][8]. Furthermore, there is growing demand by data owners' to be able to control how and where their data should be accessed, viewed, modified and distributed. Once the data is outside the perimeter of the data owner, the data owner will no longer be able to have any control over their data. A dishonest recipient can easily redistribute the data to other unauthorised users via email attachments, USB drives and in cases of documents, printing to paper. Furthermore in Cloud storage, data is usually replicated a number of times and stored in multiple locations around the world to account for higher availability.

In terms of access control, many works focus on using a promising technique called Attribute-Based Encryption (ABE) where users with certain attributes can access the data if those attributes satisfy the access control policy set out by the data owner [9][10][11][12]. However, once the data is decrypted at the user's site, that user can then redistribute the decrypted data to other users relatively easily. Digital Rights Management (DRM) [13] have allowed data owners, in particular film and music artists, to remotely control how their data should be viewed, accessed and distributed. However, many users have regarded DRM to be controversial and limiting. Some of the limitations include only opening the file using specific software or hardware, opening the file a limited number of times or only at certain times of the day and having to input a login and password every time a file is opened. These limitations only hinder the convenience of data sharing and collaboration in distributed environments. In our solution, we attempt to make the access control and monitoring as transparent as possible to the user.

- We introduce the idea of a self-controlling and self-monitoring object called SafeShare which encapsulates encrypted data and allows access to authorised users only. We extend the work of Squicciarini et al. [14] by incorporating a monitoring and auditing mechanism within the object.
- We take advantage of CP-ABE [17] and ElGamal encryption schemes [21] to develop a system called *SafeShare* which will enable secure data sharing and collaboration in the Cloud.
- Finally, we demonstrate our technology and discuss how our system can be used on a wide variety of operating systems and hardware making it feasible for everyday users to use our system.

The paper will be organised as follows. In Section 2 we discuss related work on data access control in distributed systems followed by preliminary work used to help us develop our model and protocol in Section 3. In Section

978-0-7695-5096-1/13 $31.00 © 2013 IEEE
DOI 10.1109/CSE.2013.125

IEEE computer society

4, we provide our data model, a detailed explanation of our protocol and finally follow up with a security analysis. In Section 5, our implementation details will be outlined and our experimental results analysed. Finally in Section 6, we summarise our findings and conclude the paper.

II. Related Work

Chen et al. [15] proposed bundling data with an access policy and sending this bundle to authorised users and untrusted applications. The proposed architecture called DataSafe, which enable to convert policy into hardware tags with parts of data associated with it, such as parts of documents, electronic health records, etc. which helps provide better access control. However, that data can only be accessed on DataSafe machines with special hardware, limiting the ability for user to gain access anywhere anytime. Our system can be used on any hardware and on any operating system requiring only the Java Runtime Environment.

Sundareswaran et al. [16] also bundles the data with an access policy. Additionally, a log file is also bundled with the data. Any operation the user carries out, will be appended to the log file, and this log file will be periodically sent to the Cloud. A data owner can then access the log files to check whether data is being used appropriately. This prevents man-in-the-middle attacks as well as attacks related to disassembling the bundled JAR file to read contents as logs will notify this. To prevent the tampering of log files, a hash function is used to verify integrity. The problem with this is that it does not deny control of the user to carry out illegal operations such as redistributing copies without permission. Our solution incorporates both log files and data control against illegal operations.

Squicciarini et al. [14] uses the idea of self-controlling objects (SCOs) to control how data is used. Data policies, user-created policies and jurisdiction-based policies are encoded in the SCOs along with the data. The relevant permissions are also created with the SCO. The solution uses CP-ABE [17] for policies and hence, breaking and reverse-engineering an SCO will still not retrieve plaintext data unless user is authorised. However, the user can still redistribute to other unauthorised users. The work presented in this paper is based on SCO's, however we go beyond the current solution and extend SCOs to provide better data access control.

Kayem [18] provides a solution which prevents authorised users from illegal data exchange. The solution uses an invisible digital watermark which is a hash of the encrypted data and key. However, it doesn't provide the data owner full control such as how data is to be viewed, or how many copies should be made. Burnap et al. [19] proposed a solution where parts of the data remain encrypted thoughout its lifetime and can only be decrypted if the user has access rights. However once the data is decrypted, the user can still carry out illegal operations. Kirkpatrick et al. [20] described a solution which enabled data access only to known, tursted devices. A unique device is characterised by Physically Unclonable Functions (PUFs). However, many users never stick to one machine when working and are likely to use a number of different machines. One of the distinct features of the Cloud is the ability to access data anywhere anytime, hence the need to provide more flexibility compared to this solution.

III. Preliminaries

A. CP-ABE

Ciphertext-Policy Attribute Based Encryption (CP-ABE) [17] involves encrypting data with an access control policy. A user can decrypt data if, and only if, the attributes included in his private key satisfy the access control policy. The scheme consists of the following four algorithms:

- **Setup**: Using a security parameter *L* as an input, the Setup algorithm outputs the public parameters *PK* and a master key *MK*. *PK* will be used for the encryption of data and *MK* for the generation of user attribute private keys.
- **KeyGen**: Takes as input the set of User Attributes *UA*, the Master Key *MK*, and outputs user attribute private key *UK*.
- **Encryption**: Takes as input the data *D*, an access control policy *ACP*, and public parameters *PK*, and outputs the ciphertext *E*.
- **Decryption**: Takes as input the ciphertext *E*, user attribute private key *UK*. If the set of attributes in the key *UA* satisfies *ACP* embedded in *E*, it returns *D*

B. ElGamal Encryption

ElGamal encryption, invented by T. ElGamal [21] is a public-key cryptography system. We take advantage of ElGamal Encryption in our work since the algorithm is both simple and efficient and can provide simple user revocation with low cost and overhead. There are three main steps of the ElGamal encryption algorithm:

- Initialisation: Given a prime p, a primitive root c of p, compute $b = c^x mod\ p$, where x is a randomly selected secret key. The public key is thus $\{p, b, c\}$ and private key is x.
- Encryption: Generate random value r and encrypt data m as follows:

$$\begin{aligned} E(m) &= m \cdot b^r\ mod\ p \\ &= m \cdot c^{\text{rx}}\ mod\ p \end{aligned} \tag{1}$$

  Also note: $g = c^r mod\ p$
- Decryption: This decrypts m with secret key x as follows:

$$\begin{aligned} D_x(E(m)) &= g^{\text{-x}} \cdot E(m)\ mod\ p \\ &= (c^r)^{\text{-x}} \cdot m \cdot c^{\text{rx}}\ mod\ p \\ &= c^{\text{-rx}} \cdot m \cdot c^{\text{rx}}\ mod\ p \\ &= m\ mod\ p \end{aligned} \tag{2}$$

C. SCO

Self-Controlling objects (SCOs), introduced by Squicciarini et al. [14], provides an effective way to protect data from being redistributed illegally. Data contents and access policies are encapsulated and bundled in these objects. The objects can then control who can access data and under what conditions it can be accessed. We incorporate the use of SCOs in our work to provide stronger yet flexible security protection which allows data owners to share data with many users and prevent leakage of data by dishonest users.

IV. The SafeShare System

We now introduce our *SafeShare* architecture and in particular, discuss our secure data sharing model and protocol.

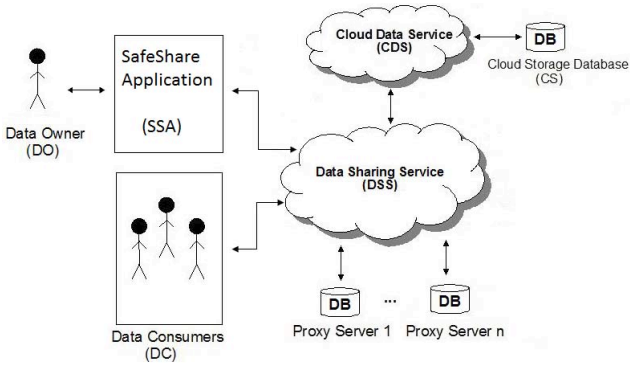

Figure 1. SafeShare Data Sharing Model

A. Model

Our secure data sharing model is highlighted in Figure 1. Table 1 summarises the role of each entity in our data sharing model. We assume throughout the rest of this paper, that the DSS is honest-but-curious in the sense that the system will follow the protocols strictly but is always curious to find out any information about stored data. Note in our data model, we assume the CDS, DSS, and Proxy Services to be semi-trusted making our solution attractive to be used in a real world scenario. We also assume the DC to be a semi-honest user since it will not be possible to completely prevent the DC from illegally redistributing data to other users since the DC can find other avenues such as taking screenshots, photographs, etc. However, we attempt to make it difficult for the curious DC to redistribute data based on commands from the software stack such as copy/paste. Hence, the DO can feel comfortable to some degree that in most cases, their data will not be misused by curious DCs. We also note that our work makes use of obfuscation techniques to help control the curious DC from illegally sharing the DOs data. Code obfuscation is a weak protection mechanism and hence the assumption of a curious DC, however we attempt to address this in future work.

DO	Data Owner	The owner of the data and decides who has access permission to the data
DC	Data Consumer	Any user who has permission to access data given by the DO
SSA	SafeShare Application	An application the DO runs to generate a SafeShare object and store to Cloud (see model)
DSS	Data Sharing Service	The service that carries out most of the data sharing functionality in the protocol (see model)
CDS	Cloud Data Service	The service that allows calls to be made to Cloud storage (see model)
CS	Cloud Storage Database	The database containing encrypted data (see model)
α	Symmetric Encryption	Symmetric encryption algorithm
δ	Symmetric Decryption	symmetric decryption algorithm
β	CP-ABE Encryption	CP-ABE encryption algorithm
θ	CP-ABE Decryption	CP-ABE decryption algorithm
γ	El-Gamal Encryption	El-Gamal encryption algorithm
τ	El-Gamal Decryption	El-Gamal decryption algorithm

Table I
Abbreviations

We make use of SCOs as used in the work of Squicciarini et al. [14] as the basis of our work and we call this object *SafeShare*. We extend upon SCOs and add a mechanism within the object that will prevent users from carrying out operations denied by the DO. Figure 2 illustrates our SafeShare object. Each SafeShare object encapsulates encrypted data contents using both symmetric key encryption and CP-ABE. We also make use of ElGamal encryption to enable efficient user revocation, which cannot be achieved by CP-ABE alone. Each SafeShare object also contains an access control policy governing which users can access data, what users can do with the data and any jurisdiction policies associated with the data. A hash is also kept of the original data for integrity purposes. Finally, a log file is kept which logs user operations on the data for auditing and accountability purposes. Each SafeShare object also contains operations such as access to data and/or make copies of the data if permitted by the data owner.

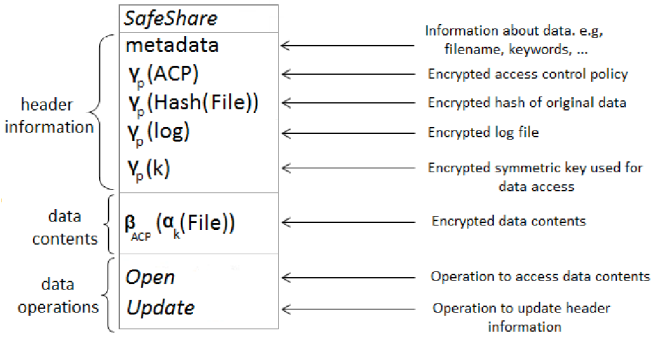

Figure 2. SafeShare object

B. Overview

We now provide a brief overview of our system. First, the DO stores his data to the Cloud by running the SSA and inputting his data and access permissions. The SSA carries out the encryption of the data and access control policies and encapsulates them into a SafeShare object along with secret keys, hash value, log, etc. The SafeShare object is then sent to the DSS and consequently the CDS for storage.

When a DC requests access to data and the DO approves, the DO calls SSA to authorise the DC. The SSA calculates new private key pieces for the DC which add up to the private key used to decrypt data. Each proxy server stores a part of the DC's key partition and the remaining key partition is sent to the DC through a medium of the DO's choosing such as USB, email, phone, etc.

Once the SafeShare object is decrypted, the DC calls the *Open()* method to access data and provides his CP-ABE private key as well as his supplied key partition. The SafeShare object verifies with the ACP what operations are allowed on the data and decrypts the data and keys if the attributes in the CP-ABE key satisfy the policy set out in the encrypted data.

Once the conditions have been met, the background process will then start which monitors for any illegal operation (see next section).

When a DO wishes to completely revoke a particular user from data access to his data, he simply calls the DSS to remove the DC's corresponding key partitions in the proxy servers.

C. Background monitoring

The background monitoring process aims to prevent a curious DC from carrying out illegal operations on the DOs data. Unless specified by the DO in the ACP, the background process does not allow the DC to modify, copy and/or paste the file to a USB or another folder for sharing.

The background process first creates a temporary folder and stores the decrypted file in the folder. The folder is made available to the DC. When the DC attempts to make a copy of the decrypted file through usual commands such as Ctrl-C on Windows, the background process checks with the ACP if the operation is allowed, and if not, immediately deletes the file in the temporary folder and leaving the clipboard empty. Similarly, when the DC is not allowed to make modifications to his file, the background process continually checks whether the file is modified. If the file is modified, the new file is immediately deleted and the user would need to run the SafeShare object again to retrieve the original file. A log file is also kept if DO requires additional control and is periodically flushed to the Cloud to prevent large data sizes on the DC's machine. The DO can then retrieve the log file from the Cloud for auditing and/or accountability purposes.

D. Protocol

We now discuss in detail our SafeShare protocol. The protocol has 5 stages namely: *data storage*, *data retrieval*, *consumer authorisation*, *authorised data access*, and *consumer revocation*.

1) Data Storage: The DO runs SSA to encapsulate their data within a SafeShare object and stores to the CS.

①	DO $\rightarrow$ SSA	INPUT: $FILE$, P, PK
②		Generate {p,b,c}, x
③		Hash($FILE$)
④		Generate log and key k
⑤		$\alpha_{\mathrm{k}}(FILE) = m$
⑥		$\beta_p(m) = M$
⑦		$\gamma_{\{\mathrm{p,b,c}\}}(P) = ACP$
⑧		$\gamma_{\{\mathrm{p,b,c}\}}(Hash(FILE)) = H$
⑨		$\gamma_{\{\mathrm{p,b,c}\}}(log) = L$
⑩		$\gamma_{\{\mathrm{p,b,c}\}}(k) = K$
⑪		Create SafeShare object SS
⑫	SSA $\rightarrow$ SS	$ACP, H, L, K, M, x_{\mathrm{n+2}}$
⑬		Obfuscate SS file
⑭	DO $\rightarrow$ DSS	credentials, SS
⑮	DSS $\rightarrow$ CDS	verifyCredentials(credentials)
⑯	CDS $\rightarrow$ DSS	u_{DO}
⑰	DSS	Generate d_{FILE}
⑱	DSS $\rightarrow$ CDS $\rightarrow$ CS	$u_{\mathrm{DO}}, d_{\mathrm{FILE}}, SS$

In the data storage stage, the DO first inputs his file, policies and public parameters from ABE to SSA on his PC ①. The application will generate public and private key pairs from the Initialisation stage of the ElGamal encryption algorithm ②. A hash of the file will then be calculated ③. A empty log file will also be generated as well as a symmetric key ④. The file will first by encrypted by symmetric key k ⑤ and then using the access control policy, the CP-ABE public key ⑥. The access control policy, data hash, log file and symmetric key will also be encrypted individually by the ElGamal public key ⑦ - ⑩. The application will then generate and store the encrypted file along with the encrypted access control policy, data hash, log file and symmetric key in an object file which we call SafeShare ⑪ - ⑫. Also, one partitioned key piece will also be stored in a variable in the source code of the SafeShare object (12) and will then be obfuscated ⑬ so that it will be extremely difficult to reverse engineer the code to find out the key piece value. This can only be found out through the running of the executable. The DO then sends the SafeShare object along with his credentials to the DSS ⑭.The DO keeps the secret key x on his machine. The DSS verifies the user credentials ⑮ - ⑯, and then generates a data id ⑰. The SafeShare object is then sent to Cloud storage ⑱. Note, we assume that the DO exists in the database with a user id. If the DO does not exist in the database, intuitively, the data will not be stored.

2) Consumer Authorisation: The DC requests the DO to access specific data stored in the CS and the DO approves.

①	DO	Calculate $x - x_{\text{n+2}}$
②		Generate $x_{\text{u1}} + ... + x_{\text{u(n)}} + x_{\text{u(n+1)}} = x - x_{\text{n+2}}$
③	DO → DSS	auth(credentials,$email$, d_{FILE}, $\{x_{\text{u1}}, ...x_{\text{u(n)}}\}$)
④	DSS → CDS	verifyCredentials(credentials)
⑤	CDS → DSS	u_{DO}
⑥	DSS → CDS	verifyUserExists(email)
⑦	CDS → DSS	u_{DC}
⑧	for (all proxy i)	
	DSS → proxy i	$\{u_{\text{DC}}, u_{\text{DO}}, d_{\text{FILE}}, x_{\text{u(i)}}\}$
⑨	DO	Generate attribute set and corresponding private key pk_{DC}
⑩	DO → DC	$pk_{\text{DC}}, x_{\text{u(n+1)}}, \{p, b, c\}$

The DO first calculates the secret key value x minus the key partition value stored in the SafeShare object ①. The DO partitions this value into $n + 1$ random pieces where n represents the number of proxy servers ②. The DO sends his credentials, the DCs identifier (ie, email), the data id and n key partitions to the DSS ③. The DSS, after verifying whether DO and DC exists ④ - ⑦, then stores the DC's key partitions to each of the proxy servers ⑧. Finally, the DO generates a private key using the CP-ABE KeyGen algorithm to generate a key for the DC which gives access to the data ⑨. The CP-ABE key along with the remaining key partition and public key is sent to the DC and he now gains access rights ⑩.

3) Data Retrieval: The DC retrieves the SafeShare object from the CS if authorised to do so.

①	DC → DSS	credentials, d_{FILE}
②	DSS → CDS	verifyCredentials(credentials)
③	CDS → DSS	u_{DO}
④	DSS → CDS	$u_{\text{DO}}, d_{\text{FILE}}$
⑤	CDS → DSS	SS
⑥	for (all proxy i)	
	DSS → proxy i	$u_{\text{DC}}, d_{\text{FILE}}$
	proxy i → DSS	$x_{\text{u(i)}}$
⑦	DSS → SS	update($x_{\text{u(i)}}$)
⑧	SS	$\tau_{x_{\text{u1}}}(ACP)$ $= \tau_{x_{\text{u1}}}(\gamma_{\{\text{p,b,c}\}}(P))$ $= (c^r, (c^r)^{-x_{\text{u1}}}.c^{\text{rx}}.P \ mod \ p)$ $= (c^r, (c^r)^{x-x_{\text{u1}}}.P \ mod \ p)$ Similarly for H, L and K
⑨	Repeat step ⑦ for all n key pieces	RP = Remaining ACP cipher: $(c^r, (c^r)^{x-x_{\text{1u}}-...-x_{\text{un}}}.P \ mod \ p)$ RH = Remaining cipher of H: $(c^r, (c^r)^{x-x_{\text{1u}}-...-x_{\text{un}}} .Hash(FILE) \ mod \ p)$ RL = Remaining cipher of L: $(c^r, (c^r)^{x-x_{\text{1u}}-...-x_{\text{nu}}}.log \ mod \ p)$ RK = Remaining cipher of key: $(c^r, (c^r)^{x-x_{\text{1u}}-...-x_{\text{nu}}}.k \ mod \ p)$
⑩	DSS → DC	SS

In this stage, the DC (or DO) downloads the SafeShare object from the Cloud ready to be accessed. The DC sends his credentials and data id to the DSS ①. The DSS verifies whether the user is legitimate ② - ③ and if so, calls the CDS to retrieve the SafeShare object ④ - ⑤. The DSS will also retrieve the DC's key partitions from the proxy servers ⑥. The DSS then calls the *update()* method of the SafeShare object and also sends the key partition value ⑦ - ⑧. The object then uses the key partition to decrypt the access policy, data hash, log file and symmetric key. The object then updates itself with these new values ⑨. The SafeShare object, containing the partially decrypted contents are then sent to the DC ⑩.

4) Authorised Data Access: The DC accesses the data in the SafeShare object provided he fulfills DO requirements.

①	DC → SS	Open($x_{\text{n+1}}, pk_{\text{DC}}$)
②	SS	$\tau_{x_{\text{u(n+2)}}}(\tau_{x_{\text{u(n+1)}}}(RP))$ $= (c^r, (c^r)^{x-x_{\text{1u}}-...-x_{\text{u(n+2)}}}.P \ mod \ p)$ $= (c^r, (c^r)^{x-x}.P \ mod \ p)$ $= (c^r, P \ mod \ p)$
③		Similarly, repeat step 2 for RH, RL and RK: P, Hash($FILE$), log, k
④		$\theta_{pk_{\text{DC}}}(\beta_P(m)) = m = \alpha_k(FILE)$ $\delta_k(m) = \delta_k(\alpha_k(FILE)) = FILE$
⑤		Create monitoring folder F
⑥	$SS \rightarrow F$	$FILE$
⑦		monitorInBackground()
⑧		IF (operation violates ACP) Delete $FILE$ immediately
⑨		IF (ACP contains doLog=true) log each operation to log file. Periodically upload log to DSS and clear log in SS

In this stage, the DC accesses the data encapsulated within the SafeShare object. The DC simply runs the *Open* function of the object, passing his stored key piece and CP-ABE private key ①. The SafeShare object then decrypts each of the metadata contents using the DC's key piece and then later the key piece stored within the source code to reveal the full metadata as well as the fully decrypted symmetric key② - ③. Note that this key value is stored in the executable binary as it is running, hence extremely difficult for the DC to ever find out this key value. The SafeShare object will use the fully decrypted symmetric key to decrypt the file. If the attributes of the private key satisfy the ACP, then the file will be decrypted to reveal the data encrypted by k. The data is then decrypted fully by k ④. The SafeShare object, after checking whether the user is authorised to access data, will generate a temporary folder and store the file in that folder ⑤ - ⑥. The SafeShare object will then monitor the file and temporary folder in the background, simulating a watchdog ⑦. If an operation violates the ACP, such as no copy or no modify, the file will be deleted immediately ⑧. If the DO

explicitly states in the ACP to enable logging, the watchdog will log any operations to the log file and update the object with the latest log. The SafeShare object will periodically flush the log file to the DSS to ensure file sizes do not exceed a maximum range ⑨.

5) Consumer Revocation: The DO revokes access rights over his data from the DC by calling the DSS.

①	DO → DSS	deleteUser(credentials, u_{DC}, d_{FILE})
②	DSS → CDS	verifyCredentials(credentials)
③	CDS → DSS	u_{DO}
④	for (all proxy i)	
	DSS → proxy i	removeKeyPiece(u_{DO}, u_{DC}, d_{FILE})
	proxy i → DSS	Remove x_{ui}

When the DO decides to revoke a user access rights to data, he simply calls the DSS to request the user to be revoked rights to the data ①. The DSS will then verify the credentials of the user ② - ③ and then provided the user exists, remove the corresponding key pieces of the user in each of the proxy databases ④. Note that the data does not need to be re-encrypted and none of the other user's will be affected since only the key pieces corresponding to the consumer is removed. All other key pieces corresponding to other consumers still remain in the proxy database. Since the data does not need to be re-encrypted nor does their need to be any key re-distribution, the model is efficient and has a runtime of O(n) where n is the number of proxy servers.

E. Security Analysis

We now analyse our model and protocol from a privacy and security perspective.

1) *Data confidentiality* – Data remains encrypted at all times whether it is in transit, within the Cloud provider or on the DC's machine. The only time the data is decrypted is when the *Open()* method is called since the class file contains the remaining key partition. Since the SafeShare object is obfuscated, the value of the key partition will be extremely difficult to reverse engineer. Without the key partition value, even if the attacking user has all other partitions, he still does not possess knowledge of the full ElGamal private key and cannot decrypt the data. Since key partitions are stored in different proxy servers (possibly modeled and implemented on different CSPs), unauthorised data access becomes extremely difficult. This is due to the fact that compromising all CSPs is almost impossible.
2) *Illegal redistribution* – The symmetric key is also encrypted with the ElGamal private key and also requires the key partition value in the obfuscated source code to retrieve the full CP-ABE key needed to decrypt data. When the object method is run, and the user has fulfilled all other requirements needed of the object, the object will decrypt the data itself and then monitor operations on the decrypted data in a background process as a watchdog. If the ACP denies redistribution, the watchdog code will prevent the DC from copying the decrypted data to another folder and sending an email attachment for instance. The watchdog is non-intrusive of user behaviour and only monitors actions on the relevant data owner's files to check for any illegal operation. It does not interfere, nor log the user actions on other files and applications.
3) *User revocation* – User revocation involves simply removing a DC's key partitions from the proxy. By doing so, the DC can never recover the full ElGamal private key. The key partition stored on the DC's machine will be rendered useless and without the other key partitions. The DC also never knows the full value of the private ElGamal key unless it is leaked by the DO. Without the full key, it is nearly impossible to decrypt the CP-ABE key needed to decrypt the data. It is also extremely difficult to decrypt the metadata information such as the ACP, hash value and log file.
4) *Auditing/Accountability* – If the DO has highly confidential data, for extra security, he can explicitly state that all operations on the data to be logged. All operations on the data will be noted by the watchdog and appended to the log file. The log file will be periodically sent to the DSS web service where the DO can access anytime anywhere to keep track of the usage of his data and to satisfy his auditing/accountability requirements.
5) *Data Integrity* – While the watchdog is running in the background, the updated hash value can also be sent to the DSS periodically. The DSS will update the previous hash value to the current one in the database. The DO can later check whether his data has been tampered with and can also hold accountable who tampered with the data. We have not focused on this aspect in great detail in this paper due to limited space.

V. Implementation and Evaluation

We now provide the implementation details of our system followed by experimental results and an evaluation.

A. Implementation

We developed our prototype of the system using the Java programming language. The DSS, CDS and Proxy servers were developed using Java, Apache Tomcat and Apache Axis2. We also used MySQL for data storage. The CP-ABE scheme, developed by Wang [22] and implements the work by Bethencourt and Sahai et al. [17]. The code was developed using the Java Pairing-Based Cryptography Library (jPBC library) [23]. To implement the SafeShare object, we made use of executable JAR files. The SSA was implemented using Java. Proguard [24] was used to obfuscate and shrink the JAR file.

B. Experimental Results

We carried out a number of performance tests on our system. In particular, we measured the overhead introduced in our SafeShare object in comparison to sharing data using only the CP-ABE scheme. We measured the performance of generating and opening of SafeShare objects. The purpose of these tests was to determine whether our system would be feasible to be used in a real-world scenario. To carry out the tests, we used a dual-core ASUS laptop with 2GB memory, 350GB storage and Windows Vista Operating System as well as a Dell XP 8500 PC with Intel Core i7 and 8GB memory running Windows 8.

For each of the tests, we used a number of files with sizes ranging from 1KB to 60MB to reflect different application requirements. The files used in the tests ranged from simple text files, to documents and also different image formats. For larger files, we used video files with different formats and 3D graphics files. For each file, the performance tests were run a number of times. The average time was then calculated and displayed in the figures below. Using the same set of files, we also measured the performance for the CP-ABE scheme of Bethencourt and Sahai et al. [17]. This allowed us to measure the performance overhead of our SafeShare object.

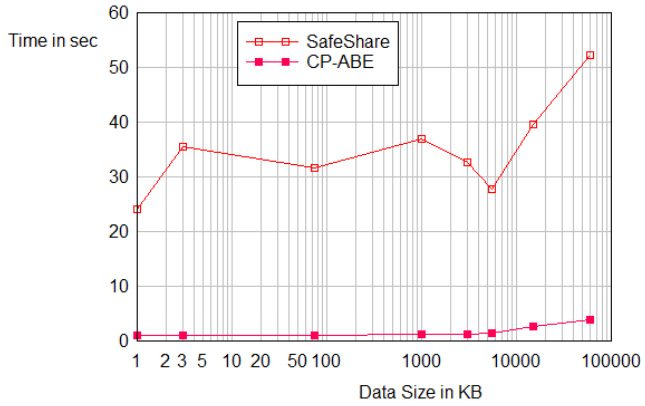

Figure 3. Data encryption overhead

Figure 3 highlights the results of our first tests. From the figure, it is clear that SafeShare objects took much longer compared to a simple CP-ABE encryption scheme. This is due to the SafeShare object having to encrypt each of the access control policy, hash and log file as well as the data itself and then package all these contents into a JAR file. As file sizes increased the data generation times increased considerably for SafeShare objects with the largest file size taking 50 seconds while the CP-ABE scheme only had a little increase in time. The slight dip in overhead of files around the 10MB mark is accounted for testing carried out in different times but is nevertheless still similar to all the file sizes below 10MB.

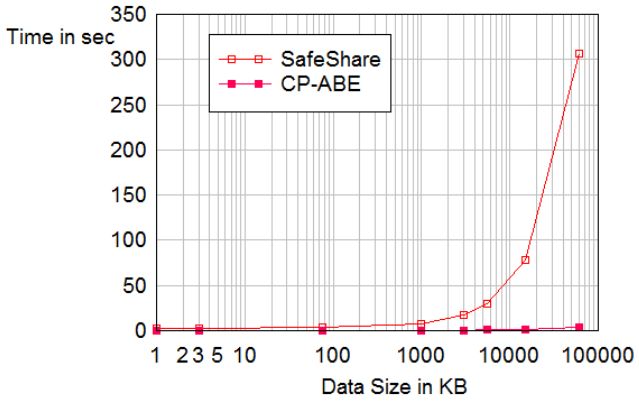

Figure 4. Data access overhead

The overhead introduced when a user carried out data access was also measured using the same set of files. Figure 4 illustrates the results of our tests where again, we measured our SafeShare scheme with the CP-ABE scheme. This is the result of running the *Open* operation of the SafeShare object. We found that for files up to 10MB in size, the overhead in access times was nearly negligible. However, once the file sizes got larger, the access times increased exponentially leaving a large overhead compared to the CP-ABE scheme. For files with sizes of 60MB, the user would have to wait approximately 5 minutes to access data each time. The access times for the CP-ABE scheme remained minimal no matter how large the file was.

C. Evaluation

From the performance tests, we found our SafeShare object to be comparably slower to encrypt and decrypt. Regarding the encryption, this is mainly to do with the generation of JAR files. Data access times increase exponentially for larger data sizes, however as processing power increases and the generation of JAR files optimises, this will improve access times in the near future. In our current solution, users may be willing to wait a little longer for highly confidential data to generate and access data. For instance, a business user may be willing to wait for a little more than a minute to access 15MB of highly confidential paperwork. This makes our solution highly attractive to users who are more concerned over privacy compared to performance times.

VI. Conclusions

There is a growing demand for big data sharing and collaboration in distributed environments such as the Cloud. One of the main issues with big data sharing in such environments is the privacy and security of information. In particular, the issue of illegal redistribution of data by dishonest users and the need to keep data highly confidential

while respecting the policies set out by the data owner. In this paper, we have developed a secure data sharing system called *SafeShare*, which allows highly confidential data sharing while preventing illegal redistribution of data by dishonest users. We also developed a set of security models and protocols and carried out a security analysis on our protocol.

REFERENCES

[1] M. Healey (2010): Why IT Needs To Push Data Sharing Efforts. Information Week. Source: http://www.informationweek.com/services/integration/why-it-needs-to-push-data-sharing-effort/225700544.

[2] D. A. Riley (2010): Using google wave and docs for group collaboration. Library Hi Tech Newss, Vol. 27 Iss: 4/5: 12 - 14

[3] R. Wu (2012): Secure sharing of electronic medical records in cloud computing. Arizona State University. ProQuest Dissertations and Theses.

[4] A. Gellin (2012): Facebook's benefits make it worthwhile. Buffalo News (Buffalo NY). Dialog LLC. 2012. Retrieved May 07, 2013 from HighBeam Research: http://www.highbeam.com/doc/1P2-30776177.html

[5] D. Bender (2012): Privacy and Security Issues in Cloud Computing. Computer & Internet Lawyer: 1-15.

[6] H. Judith, B. Robin, K. Marcia, H. Fern (2009): Cloud Computing for Dummies. For Dummies.

[7] S. SeongHan, K. Kobara, H. Imai (2011): A Secure Public Cloud Storage System. Internet Technology and Secured Transactions(ICITST), 2011 International Conference: 103-109

[8] M. Zhou, R. Zhang, W. Xie, W. Qian, A. Zhou (2010): Security and Privacy in Cloud Computing: A Survey. Semantics Knowledge and Grid (SKG), 2010 Sixth International Conference: 105-112

[9] Y. Yang, Y. Zhang: A Generic Scheme for Secure Data Sharing in Cloud. Parallel Processing Workshops, 2011 40th International Conference: 145 - 153

[10] S. Tu, S. Niu, H. Li, Y. Xiao-ming, M. Li (2012): Fine-grained Access Control and Revocation for Sharing Data on Clouds. Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International: 2146-2155

[11] S. Yu, C. Wang, K. Ren, W. Lou (2010): Achieving Secure, Scalable, and Fine-grained Data Access Control in Cloud Computing. 2010 Proceedings IEEE: 1-9

[12] M. Li, S. Yu, Y. Zheng, K. Ren, W. Lou (2013): Scalable and Secure Sharing of Personal Health Records in Cloud Computing Using Attribute-Based Encryption. Parallel and Distributed Systems, IEEE Transactions: 131-143.

[13] Digital Rights Management. Wikipedia. Source: http://en.wikipedia.org/wiki/Digital_rights_management

[14] A. Squicciarini, G. Petracca, E. Bertino (2013): Adaptive Data Protection in Distributed Systems. Third ACM Conference on Data and Application Security and Privacy (CODASPY), February 2013: 365 - 376

[15] Chen Y.; Jamkhedkar P.A.; Lee R.B. (2012): A Software-Hardware Architecture for Self-Protecting Data. In Proceedings of the 19th ACM Conference on Computer and Communications Security, October 2012: 14 - 27

[16] Sundareswaran, S; Squicciarini, A.C.; Lin, D (Jul/Aug 2012): Ensuring Distributed Accountability for Data Sharing in the Cloud. IEEE Transactions on Dependable and Secure Computing, Vol. 9, No. 4: 556 - 568

[17] Bethencourt, J.; Sahai, A.; Waters, B. (2007): Ciphertext-Policy Attribute-Based Encryption. Security and Privacy, IEEE Symposium: 321 - 334

[18] Kayem, A.V.D.M. (2010): On monitoring information flow of outsourced data. Information Security for South Africa (ISSA), 2010: 1-8

[19] Burnap, P.; Hilton, J. (2009): Self Protecting Data for De-perimeterised Information Sharing. Digital Society, 2009. ICDS '09. Third International Conference: 65-70

[20] Kirkpatrick M.S.; Kerr S. (2011): Enforcing physically restricted access control for remote data. ACM conference on Data and application security and privacy (CODASPY '11): 203-212.

[21] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. Advances cryptology, 1985: 469 - 472

[22] Wang J. CP-ABE Java Implementation using jPBC. Source: http://wakemecn.com/cpabe/

[23] The Java Pairing Based Cryptography Library (jPBC) website. Source: http://gas.dia.unisa.it/projects/jpbc/

[24] ProGuard JAR Shrinker and Obfuscator. Source: http://proguard.sourceforge.net/

The author has requested enhancement of the downloaded file. All in-text references underlined in blue are linked to publications on ResearchGate.