Policy Carrying Data for Enhancing User Privacy in Cloud Based Systems

by

Sameer Balasubrahmanyam

A Project Report Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science in Computer Science

Supervised by

Dr. Rajendra K. Raj

Department of Computer Science

B. Thomas Golisano College of Computing and Information Sciences Rochester Institute of Technology Rochester, New York

December 2016
User privacy continues to remain a threat on cloud based systems. This paper proposes a technique for protecting user privacy in cloud based systems known as Policy Carrying Data. Policy Carrying Data or PCD enables the user to attach Terms of Service to the data before uploading it to the cloud environment. Unlike most privacy enhancing models, PCD does not obfuscate data from the cloud for the entire life time of the data, instead it obfuscates data only until the cloud service provider agrees to comply with the ToS attached by the user. In this context PCD is achieved through a cryptographic technique known as Ciphertext Policy Attribute-Based Encryption.

This paper introduces a model where the users can embed ToS in the form of attributes in JPEG EX-IF metadata. The image then runs through a tool that generates the ciphertext image which is then uploaded to the cloud environment. The cloud service provider gains access to the plaintext image only if it has a corresponding attribute key which is generated by the same certificate authority. In order to gain access to that key, the cloud service provider has to agree to comply with the policy to the certificate authority. These policies are legally binding and any decryption operations support non repudiation. In the event of a policy violation a cloud service provider cannot deny the violation.
# Contents

Abstract ................................................................. ii

1 Introduction .......................................................... 1

2 Related Work ......................................................... 4

3 Background ............................................................ 7
   3.1 Attribute Based Encryption ................................... 7
   3.2 Terms of Service .............................................. 8

4 Design .......................................................................... 10

5 Implementation .......................................................... 12
   5.0.1 PCD Certificate Authority ................................. 12
   5.0.2 ABE Encryption Tool ....................................... 12
   5.0.3 PCD Image Sharing ........................................ 13

6 Analysis ........................................................................ 14
   6.1 User Involvement .............................................. 14
   6.2 Performance ..................................................... 14
   6.3 Threat Model ................................................... 16

7 Conclusions ............................................................... 19
   7.1 Current Status .................................................. 19
   7.2 Future Work .................................................... 19

Bibliography ............................................................... 21

A UML Diagrams .......................................................... 23
   A.1 Sequence Diagram ........................................... 23
   A.2 Activity Diagram .............................................. 24
B Code Listing .................................................. 26

C User Manual ................................................ 27
C.1 ABE Encryptor ......................................... 27
   C.1.1 Requirements .................................... 27
   C.1.2 Installation ..................................... 27
   C.1.3 Running ......................................... 28
C.2 PCD CA .................................................. 28
   C.2.1 Requirements .................................... 28
   C.2.2 Installation & Running ......................... 28
C.3 PCD Image Share ....................................... 28
   C.3.1 Requirements .................................... 28
   C.3.2 Installation ..................................... 29
   C.3.3 Running ......................................... 29
List of Tables

6.1  Time taken by users to use ABE Encryptor App  . . . . . . . .  18
List of Figures

4.1 PCD Architecture .......................................................... 10
6.1 ABE Vs AES Encryption Time ............................................. 15
6.2 ABE Vs AES Decryption Time ............................................. 16
6.3 ABE Encryption/Decryption time Vs Number of attributes ........ 17
A.1 PCD sequence diagram .................................................... 23
A.2 Cloud Service Provider Upload Handler ............................... 25
Chapter 1

Introduction

Cloud services have revolutionized the way we think of traditional software and software based services. Offloading computations to the cloud can help deploy cheaper, more scalable solutions easily [7]. Cloud environments such as platform as a service (Paas) like Amazon Web Services, Microsoft Azure have enabled businesses to deploy scalable, data driven applications that are available to millions of users within minutes. Traditional software that was installed on personal computers is moving into the cloud with the rise of software as a services (SaaS) [3]. With this change come in a plethora of privacy concerns [5]. This is because the data that lived on personal computers now lives on the cloud environment and the entity hosting the services and data dictate how they should be used.

Cloud service providers offer free or subsidized services in exchange for user data. Instead of charging the consumers, the cloud service providers monetize on personal user information. User info is often sold to third party advertisers [10]. Personal information that advertisers are interested in typically are GPS location, purchase history, browser history etc. The information from this data is used to serve personalized advertisements to the user. Examples of other personal information include emails, chat messages, pictures, personal files, documents.

Most of the solutions proposed to solve the growing privacy concerns are based on two fundamental ideas: obfuscating data from the cloud [2] or not using the cloud all together. In the former solution data is encrypted with a symmetric key before its uploaded to the cloud. The user keeps this key private and hence the cloud service provider never gains access to the data. However this greatly limits the capabilities of the cloud. The cloud
cannot perform any computation on the cipher text data and it can merely store the cipher text blob. In the later technique, the user avoids using the cloud service by setting up a private server. An example of this using OwnCloud instead of Dropbox or Google Drive, where personal files are stored on a home computer but is accessible over the internet. The solution proposed in this paper helps preserve privacy and uses cloud environments while harnessing the computation power of the cloud.

There are a set of challenges associated with building such a privacy preserving system that would be used in commercial applications. To start with, it should not make it difficult for the end user to use the system. People lean towards ease of use over security. It should be easy for the cloud service providers to on board onto the system. Cloud service providers must make minimal changes to their existing infrastructure to adopt the system. It should not only protect data uploaded by the user but also protect derived data from the original data. The proposed solution must be practical and should not favor users or cloud service providers as this affects adoption.

In this paper we propose an alternative solution called Policy Carrying Data. The solution encompasses around letting users attach terms of service (ToS) to data before uploading it to the cloud environment. The way cloud service providers currently operate is that the user is presented with a set of terms and conditions that entitles the service provider to monetize on user data. Because these terms of service are often abstruse and full of legal jargon, the average user never reads through the 7000 odd word privacy policy. The model in this paper lets the data owner dictate how the data should be used.

The model specifically deals with protecting JPEG images but we will explain how it can be extended to include all file types. The user can embed policies into the image meta data. Our ABE Encryptor will extract those policies and encrypt the image using Attribute Based Encryption. The cipher text generated is then uploaded to the cloud service environment. During the one time setup phase, the cloud service provider registers with the certificate authority the policies it complies to and the certificate authority generates an attribute key for the cloud service provider. If the user defined attributes match the cloud
service provider’s attribute set, the attribute key will successfully decrypt the cipher text. However if the attributes do not match the cloud, the service provider will never get access to the plain text data. PCD is a powerful tool that helps users and cloud service providers negotiate over terms of service. For example if a user wishes to attach a restrictive policy that does not allow the service provider to use the data for advertising then the user can opt in to pay for the service. However if the user is comfortable with a liberal policy that allows the service provider to monetize then the service provider may offer the service for free.

In this paper we first look at some related work in the field of Policy Carrying Data, we then look at how attribute based encryption is used to help implement Policy Carrying Data. Then we describe the proof of concept implementation of the proposed solution, along with some evaluations of its performance and security before discussing its limitations and score for future work.
Chapter 2

Related Work

Even though there has been a lot of work in the field of protecting privacy in cloud based systems, in this paper we will focus specifically on the technique of policy carrying data. Due to the fact that there are many ways to incorporate policy carrying data in a cloud based system it is crucial to look at other work that has been done in this field.

In the paper titled Protecting Outsourced Data Privacy with Lifelong Policy Carrying [9] X. Wang et al. propose a system that accomplishes similar goals but with a different approach. The system proposed by X. Wang et al. combines logical policies with the data and also incorporates a remote access control system that parses these policies before offloading them to the cloud or third party environment. The system protects everyone from the end user to the remote service provider. The proof of concept implementation of the above system is known as Logical Data Propagation and Access Control or LDPAC. The LDPAC system itself consists of 3 major components: Translator, LDPAC verifier, app container. The translator is responsible for combining the logical policies with the sensitive data before transferring the data to the cloud environment. Once the data reaches the environment any accesses made to the data by the application are routed to the LDPAC verifier. The LDPAC verifier evaluates access based on the runtime parameters and the application attributes. The LDPAC verifier is responsible to determine if access must be granted or denied to each request. The app container abstracts the interaction between LDPAC verifier and the application. Because the application lives within the container all data access requests are routed through the LDPAC verifier.
Unlike our proposed system, LDPAC operates at the operating system level and is transparent to the application. Any data access requests to the file system are routed through the LDPAC module. The policy structure described in their work is similar to the ones described in this paper i.e they are a set of attributes value pairs chained with conjunction or disjunction operators. Compliance is guaranteed through the LDPAC system however in case the LDPAC verifier gets compromised, compliance is not guaranteed via third party; in our model compliance is guaranteed through the PCD Certificate Authority.

However the system does have limitations. A system like LDPAC is not interchangeable across different application and platforms. It is difficult to audit and create a standard out of such a system, as different systems will have varied requirements. Secure coding flaws in the LDPAC may lead to data leaks, unauthorized access and corrupt history. The model also includes the notion of an app container which abstracts system calls such as network functions (send, recv) or file system calls (write, read) into a container to route requests to the LDPAC module. This is problematic from the point of view of maintainability and the app container would have to be updated if there is a breaking change in the API which is uncommon but possible.

The paper titled Policy-Carrying Data: A Privacy Abstraction for Attaching Terms of Service to Mobile Data [8] describes a model for protecting mobile data in cloud environments. The model created by Stefan Saroiu et al. in this paper uses Attribute Based Encryption for implementing PCD [8]. The two techniques used to bind policies to data described in this paper are JSON and RESTful services. Using JSON, policies can be added as key value pairs, while in RESTful services the policies can be passed through HTTP headers. The entire model consists of a user encrypting the data with the policies and a certificate authority’s public key before uploading it to the cloud. The cloud service provider registers with the certificate authority with the policies it complies with. The certificate authority then generates an attribute key that is used by the cloud service provider to decrypt the data uploaded by the user if the policies match.

The paper does not implement a working model of PCD, what it does is contribute an
implementation of CP-ABE encryption library [4]. The implementation continues to be one of the paper's greatest contributions and is released under the liberal GPL. The paper does not cover problems of on-boarding or scaling cloud-based systems with PCD.
Chapter 3

Background

3.1 Attribute Based Encryption

In traditional public key cryptography, data is encrypted with the receivers public key. The data cannot be shared by multiple users, this is only possible with symmetric key cryptography. However the same symmetric key is shared with multiple receivers which is considered a bad practice. Attribute Based Encryption builds upon this pretext. The concept of ABE was first introduced by Amit Sahani and Brent Waters in Fuzzy Identity Based Encryption [6]. In Identity Based Encryption the decryption key is generated based on a set of user attributes. In the context of access control this may consist of user attributes such access level, user type, state of the system, type of the data etc.

In ABE, encryption is performed using a set of attributes, while decryption is performed by a private key generated over a set of user attributes. If the set of attributes match, decryption is successful. The key is independent of the data and a same key can be used to decrypt different data. The set of attributes may differ for users because the set of attributes used for encryption may contain conditionals such as conjunction, disjunction and n of m thresholds.

ABE encryption and decryption is a 4 step algorithm.

- **Key Generation:** A set of private and public key is generated by the setup operation. Let these keys be denoted by $K_{\text{pub}}$ and $K_{\text{priv}}$. A decryption denoted by $K_{\text{attr}}$ is then generated by $K_{\text{attr}} = \text{key\_gen}(\text{Attributes, } K_{\text{prv}})$
• **Key Sharing:** The $K_{pub}$ is shared with both the encryption and decryption party. The $K_{attr}$ is shared with the decryption party.

• **Encryption:** The plain text data $P$ is encrypted using set of attributes and the $K_{pub}$ to generate cipher text $C$. $C = \text{enc}(\text{Attributes, } K_{pub}, P)$

• **Decryption:** The plain text data $P$ is obtained from the cipher text $P = \text{dec}(C, K_{attr}, K_{pub})$

### 3.2 Terms of Service

Terms of service in this context consist of key value attributes passed to the ABE encryption operation. Multiple key value pairs can be combined with conjunction or disjunction.

Example of such a policy can be.

```plaintext
allow_face_recognition: false and
   allow_indexing: false
```

They may also be combined using $n$ of $m$ threshold for example

```plaintext
allow_face_recognition: false
   allow_indexing: false 1of2
```

It is up to the user to dictate the policies, and these policies could determine various aspects of how the data is used. For example the policy could be hardware binding. The following policy mandates the use of a Hardware Security Module or Trusted Platform Module.

```plaintext
... require_TPM: true ...
```

The policy could be software binding for example the data could require to be encrypted by a specific cryptographic algorithm or require a minimum level of an operating system.

```plaintext
encryption: aes256 and
   min_os_ver: win7
```
The policy could also control the geographic location where the data should be stored.

data_center_region:US

The policy could control who gets access to the data. For example a data owner could prevent the service provider from sharing it with third parties.

share_with_third_party:false and 
share_withAdvertiser:false

The policy could dictate when the data can be used or how long it could be stored.

retention_period=3m and 
access_grant_date=11/15/2016

In theory, PCD enables users to attach arbitrary ToS to their data. However this is not very practical because cloud service providers will not have the appropriate keys for decryption. PCD should instead create standardized policies that users can opt to use. This is similar to how software licenses operate. Developers usually bundle their software with standard licenses such as GNU General Public License, BSD License, Apache License, MIT License etc. instead of writing their own. Based on the restrictiveness of the policy the cloud service provider may invoice the storage accordingly.
Chapter 4

Design

The organization of the model is shown in Figure 4.1, it consists of the following components.

- **Certificate Authority**: In the setup phase the CA generates a pair of public and private key. This public key is shared with the user as well as the cloud service provider. The private key is kept secret for all further operations. During the one time setup phase the cloud service provider presents its set of policies and other required personal
information to prove its authenticity. Once the service is verified the CA generates an attribute key for the cloud service provider which is then used to decrypt the cipher text uploaded by the user.

- **Pre Internet Encryption:** In this component the data is encrypted with the CA public key and the policy. The generated cipher text is then uploaded to the cloud. The core concept is that the cloud should not get access to the plain text data if it does not comply with the policy, and that any data that leaves the client is cipher text. This is achieved using Attribute Based Encryption where cipher text is a function of plain text image, CA public key and policy.

- **Cloud Service Provider:** In this model the cloud service provider initially registers with the CA with the set of policies it agrees to comply with, the CA then generates a attribute key that can be used to decrypt only the data that was encrypted with the corresponding policy.
Chapter 5

Implementation

The goal of this implementation was to create a proof of concept end to end infrastructure to help create PCD aware applications. We also create a PCD aware image sharing service to consume the infrastructure. The implementation consists of the following components.

5.0.1 PCD Certificate Authority

PCD certificate authority is implemented as a web service and it enables cloud service providers to register themselves with the set of attributes they agree to comply with. The PCD CA maintains a record of the service providers and the issued keys. The certificate authority also provides its public key that is used by both the end user and the service providers to ensure that they are talking with the same certificate authority.

The certificate authority was implemented using Java and Java Play Framework. The service can be deployed on Windows, Mac OSX or Linux as the Java and Play Framework because it is cross platform.

5.0.2 ABE Encryption Tool

ABE Encryption tool is the software that the user uses to encrypt the data with the policy before uploading it to the cloud service. The tool has a GUI and is built with the Electron framework using NodeJS and JavaScript. The app is cross platform and can run on Windows, Mac OSX and Linux. The inputs of the application include plain text file, Certificate Authority public key and set of policies. The policy may be explicitly specified or set to be embedded into the JPEG meta data in the Copyright field. There is a toggle switch
that helps the user switch between embedded policy mode or explicit policy mode. Once the input parameters are set, the user can click on the encrypt button and this generates an encrypted version of the file that can be uploaded to the PCD aware cloud service. Even though the front end for the application is written in JavaScript the encryption functions are called from a Java jar using a custom bridge API.

5.0.3 PCD Image Sharing

In order to prove out the PCD model and test the PCD infrastructure, our implementation includes a PCD Image Sharing service. It enables users to upload and share images with policy. The service has two buckets of users: free and paid. Depending on the policy restrictiveness the service will classify the resource accordingly. The service has two attribute keys for each of the policies. When the user uploads the cipher text image the service cycles through each of the keys. Once the decryption is successful, the file store records the policy type for each image. Upon viewing the image there is a visual indicator tag that displays if the image was classified as free or paid.

The service is written in JavaScript using NodeJS, uses MongoDB for file store and database and can be deployed on Windows, Mac OSX or Linux.
Chapter 6

Analysis

6.1 User Involvement

One of the goal of the project was to make the system as easy as possible for the user. Therefore we did some usability testing. Three different users were asked to perform the following steps:

2. Select PCD CA public key file that was already present on the machine
3. Select Input file with policy
4. Retrieve the resulting cipher text files

The results are tabulated in 6.1, the second run was faster than the first run and the total work flow was under 10 seconds of additional time required. We believe more automated tools and menus built into Operating System context menus can greatly cut down this time further.

6.2 Performance

Before we dive into the benchmarks it is important to explain what we benchmark and why. We compare Attribute Based Encryption operations with AES operations. It is important to note that ABE is not a replacement for AES or any symmetric key cryptographic system.
But it does give us a good baseline for comparison because AES is a very common cryptographic system that is employed in almost every cloud system. However it is important to note that the library used in our implementation may not be optimized and is currently in a very nascent stage. The AES implementation that was used for this benchmark was the one built into JDK Security. All benchmarks were done on an AMD A-10 5750M with all 4 cores running at 2.5GHZ with 8GB DDR3 ram. The operating system was Windows 10 64bit. All encryption operations include time taken by the program to read the plain text input file, perform the encryption and write the cipher text file back to the disk. Similarly decryption is the time taken by the program to read the cipher text file, decrypt the data and write the plain text back to the disk.

![Figure 6.1: ABE Vs AES Encryption Time](image)

Figure 6.1 compares ABE encryption with AES encryption. The policy used for encryption consisted of 2 attributes. We can see that ABE has a marginal overhead compared to AES. And this overhead is constant for the most part.

Figure 6.2 compares ABE decryption with AES decryption. The same policy was used.
Figure 6.2: ABE Vs AES Decryption Time

ABE had a minor overhead compared to AES, however as the file size increased ABE encryption was faster than AES. This is because AES is a block chain cipher and slows down as the file size increases.

Figure 6.3 compares ABE encryption and decryption growth with the number of attributes. We saw that the time required for encryption and decryption did increase with the increase in number of attributes but the growth was not exponential. Even for a 10 attribute value policy, encryption and decryption operations were fairly within practical limits. Decryption was always faster than encryption.

6.3 Threat Model

Every security model has its pros and cons. It is crucial to evaluate the types of attack our model does and does not protect the user against. The model does not protect against any sort of physical attacks. Physical theft of servers or hard drives may leak keys and data.
A common attack that exists in PCD systems is a policy removal attack. This is when an attacker targets to steal the plain text data that lives without the policy. This can be done in two ways: attack the plain text on disk and attacking the plain text in memory. Our model protects us against the former as plain text is never retained on the disk. For every single data access request the cipher text is decrypted and the plain text is immediately destroyed. However this model does not explicitly protect us against a policy washing memory attack. We rely on process isolation and address space layout randomization feature of the operating system that make it very difficult to execute such an attack.

Policy brute forcing is another attack possible on such a system. This attack assumes that the cipher text data is compromised from the cloud service. An attacker can now brute force all possible attribute value pairs and gain access to the plain text. However this attack is not very practical as the certificate authority alone can generate keys. The certificate authority mandates business and personal information before it generates keys. It could optionally rate limit key generation if it detects malicious activity.
Table 6.1: Time taken by users to use ABE Encryptor App

<table>
<thead>
<tr>
<th>User</th>
<th>Platform</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>User 1</td>
<td>Windows</td>
<td>6590</td>
</tr>
<tr>
<td>User 1</td>
<td>Windows</td>
<td>5830</td>
</tr>
<tr>
<td>User 2</td>
<td>Ubuntu</td>
<td>7725</td>
</tr>
<tr>
<td>User 2</td>
<td>Ubuntu</td>
<td>6405</td>
</tr>
<tr>
<td>User 3</td>
<td>Mac OSX</td>
<td>9560</td>
</tr>
<tr>
<td>User 3</td>
<td>Mac OSX</td>
<td>9001</td>
</tr>
</tbody>
</table>
Chapter 7

Conclusions

7.1 Current Status

The current implementation is a good starting point but it remains a proof of concept implementation. More vigorous design and standards are required for production quality implementation. The ABE Encrpytor application can be made easier to use by integrating it within the system file manager. For example having the ability to right click on a file for encryption.

The current implementation of the PCD Image Share was built as a PCD aware service from scratch. The effort required to migrate existing cloud applications may be significant depending on the applications. Our model does not accommodate an boarding process.

7.2 Future Work

PCD and ABE are in very early stages of development and still require a significant amount of work.

As mentioned earlier, Cipher text Policy Attribute Based Encryption should be standardized and have reference implementations that are open source. These libraries should be audited regularly especially in early stages of development.

Most modern processors incorporate an atomic AES operation within the native CPU instruction set. This helps speed the algorithm as a single atomic operation can be performed in a single CPU cycle. Such a development for Attribute Based Encryption would be ground breaking in this field.
Reusable policies should be documented and categorized. This helps users understand the policies they attach to their data. This is similar to how content providers bundle licenses.

As mentioned earlier, PCD does not provide a detection mechanism. A detection mechanism would certainly benefit PCD users but the system should not have proprietary access control and monitoring because this raises secondary privacy concerns similar to Digital Rights Management (DRM) systems [1].
Bibliography


Appendix A

UML Diagrams

A.1 Sequence Diagram

Figure A.1: PCD sequence diagram
A.2 Activity Diagram
Figure A.2: Cloud Service Provider Upload Handler
Appendix B

Code Listing
Appendix C

User Manual

C.1 ABE Encryptor

C.1.1 Requirements

Operating System:

- Windows 7 or above
- Ubuntu 12.04 or above
- Mac OSX 10.8 or above

Dependencies:

- NodeJS https://nodejs.org
- Google Chrome https://www.google.com/chrome
- Node Package Manager https://www.npmjs.com

C.1.2 Installation

Run the following command in the terminal

    cd ABE_Encryptor && npm install
C.1.3 Running

Run the following command in the terminal

```
cd ABE_Encryptor && npm start
```

C.2 PCD CA

C.2.1 Requirements

Operating System:

- Windows 7 or above
- Ubuntu 12.04 or above
- Mac OSX 10.8 or above

Dependencies:

- Lightbend Activator [https://www.lightbend.com/activator](https://www.lightbend.com/activator)

C.2.2 Installation & Running

Run the following command in the terminal

```
cd pcd-ca && activator run
```

Then visit [http://localhost:8888](http://localhost:8888) in your browser

C.3 PCD Image Share

C.3.1 Requirements

Operating System:
• Windows 7 or above

• Ubuntu 12.04 or above

• Mac OSX 10.8 or above

**Dependencies:**

• NodeJS [https://nodejs.org](https://nodejs.org)

• Node Package Manager [https://www.npmjs.com](https://www.npmjs.com)

• MongoDB [https://www.mongodb.com](https://www.mongodb.com)

C.3.2 **Installation**

Run the following command in the terminal

```
cd pcd-img-share && npm install
```

C.3.3 **Running**

Run the following command in the terminal

```
cd pcd-img-share && node app.js
```

Then visit **localhost:3000** in your browser