SafeProtect: Controlled Data Sharing with User-Defined Policies in Cloud-based Collaborative Environment

Danan Thilakanathan\textsuperscript{1,2}, Shiping Chen\textsuperscript{2}, Surya Nepal\textsuperscript{2} and Rafael Calvo\textsuperscript{1}

\textsuperscript{1}School of Electrical and Information Engineering, The University of Sydney, NSW, Australia
\textsuperscript{2}Digital Productivity Flagship, CSIRO, Marsfield, NSW, Australia

There are many Cloud-based applications consumed by users which encourage data sharing with not only peers, but also new friends and collaborators. Data is increasingly being stored outside the confines of the data owner's machine with little knowledge to the data owner, how and where the data is being stored and used. Hence, there is a strong need for the data owner to have stronger control over their data, similar to the level of control they possess when the data is stored on their own machine. For instance, when a data owner shares a secret file with a friend, he cannot guarantee what his friend will do with the data. In this paper, we attempt to address this problem by monitoring and preventing unauthorised operations by the data consumer. We present a solution called SafeProtect which bundles the data owners data and policy, based on XACML, in an object. SafeProtect enforces the policies set out by the data owner by communicating with the SaaS applications to disable certain commands and/or run a background process monitor for auditability/accountability purposes. We define a protocol that will enable secure data sharing in the Cloud and leverage the use of the Trusted Extension Device (TED) for authentication purposes.

\textit{Index Terms}—access control, cloud, distributed computer networks, privacy, security, policy, monitoring.

I. INTRODUCTION

In recent times, there has been an explosive growth in data sharing and collaboration by both social users and enterprise users alike [1]. This is further aided by popular platforms such as YouTube, Facebook and Twitter, which demonstrate the strong impact data sharing has on today's generation. The growth of data sharing in recent times has provided an abundance of information and resources for all types of users, providing benefits in education and research [3] [4] [5] [6], medicine [7] and entertainment [8] to name a few. This has improved the quality of life for a lot of people and allows them to express their views, talents as well as connect with and even do business with people from the other side of the world. This level of accessibility to data and information was unimaginable a decade ago.

With this strong need for data sharing in the Cloud, it is imperative that privacy and security needs of the data owners are addressed [9] [10]. When privacy and security of data is breached, data owners lose trust with sharing their information [17] [18] [19] [20]. This can also impact businesses when a trade secret is leaked. While there are many literature on securing data in the Cloud [11] [12] [13] [1] [41], little work has focused on giving the data owner an even greater level of control to their data. One particular growing concern of data owners is the data access control problem. Once the data owner has provided rights to a data consumer to access data, he can no longer guarantee that the data consumer will safeguard the data to the level of degree expected by the data owner. The data consumer can redistribute the data to their peers and colleagues without the knowledge of the data owner. This is one of the main reasons why data owners are apprehensive about sharing data with others, especially complete strangers.

There needs to be a way for data owner’s to have full control over their data. The data owner must be able to specify how their data is to be used and what operations are allowed by which specific data consumers. For instance, a patient may share health data with doctors and medical professionals. The patient should be able to specify a policy that allows some doctors to modify the health data (e.g., his local doctor) but restrict other doctors to only view it and not redistribute it to other doctors without knowledge of the patient. If the user performs operations beyond the policy, it would be treated as unauthorised as long as it is covered by the terms and conditions and users are made aware of it.

Moreover, data sharing in the Cloud places a considerable burden on the Cloud servers as well as on data owners to manage data consumers and their access to data. One such problem is key management. When a data owner wishes to revoke a particular data consumer from access to their data, the data owner would traditionally have to re-encrypt their data and re-distribute new keys to the remaining users. This places a tremendous burden on data owners especially with large groups. Some solutions have also explored leaving the re-encryption to the Cloud which may prove costly [1]. Bring Your Own Device (BYOD) is a fast growing trend encouraging employees to bring their own devices such as smartphones, tablets and laptops to access sensitive enterprise information. BYOD can help address the key management problem since a unique key can be tied to a particular consumer’s device. However, BYOD is not enough to enable secure data sharing in the Cloud and is also susceptible to privacy and security attacks [14] [15]. For example, if the consumer’s device contains malware, it can steal enterprise information and transmit to attackers. Also the device may leave traces of very sensitive information later on.

Furthermore, data must also be protected over the wire and in Cloud storage servers [16]. Malicious insiders provide the
most potential threat to data owner’s data since they have first point of contact to the data. A corrupt database administrator may access the database and reveal the data owner’s data. Cloud Service Providers (CSPs) may also sell data to third parties in order to gain profit. There have been many examples of this in the real-world [21] [22] [23]. Hence the data must remain protected at all times.

A. Motivation

Current applications, such as Microsoft Word, lack the ability to give data owner fine-grained access control over their data. Due to the explosive growth in data sharing in recent times, applications need to take into consideration privacy and security. Newer versions of Microsoft Word, for instance, provide capabilities to protect and encrypt the data, however, it doesn’t allow data owners to specify policies on how the data should be used. Applications can encourage this growth in data sharing by allowing data owners to specify a policy on who can access their document, what data consumers are allowed or not allowed to do with their document, and where the document can be accessed. By doing so, data owners gain a greater level of trust in data sharing with the knowledge that their data will not be misused or leaked without their permission. This allows only data owners and authorised data consumers to access the data irrespective of where the application is installed and how the data is accessed. Creating such a feature in current applications provides benefits to not only the data owners, but also the application companies through higher sales and revenue. In our solution, we envision applications enforcing policies set out by the data owner through a TPM device to help achieve stronger privacy and security of the data owner’s data.

We consider a scenario where a patient Bob, wishes to share his health data to several doctors and medical professionals. Bob would like to restrict Dr. Alice to only view his health data but wouldn’t mind sharing all his health information to his local GP Dr. Tim. Furthermore, Bob doesn’t want Dr. Alice to share the health data with her peers and colleagues. There is currently no way he can completely prevent Dr. Alice from doing this. He can, however, attempt to make it difficult by providing the health information using a SaaS-based application such as Google Docs, but she can still save out a copy or even copy/paste the entire document.

B. Our contributions

In this paper, we address both the key management problem as well as the data owner access problem by making the following contributions:

- We build upon our previous paper [39] and propose a novel scheme that will allow private and secure data sharing in the Cloud using a physically-based TPM device called the Trust Extension Device (TED).

C. Organisation of the paper

The paper will be organised as follows. In Section 2, we discuss related work. In Section 3, we briefly discuss our preliminaries such as TED, ElGamal encryption, SafeProtect and policy language. In Section 4, we describe the security requirements of our system. We then introduce our scheme in Section 5. We describe the implementation of our scheme in Section 5. In Section 6, we provide a security analysis of our scheme. In Section 7, we describe the implementation of our scheme and discuss the performance of our scheme and evaluate the feasibility of our scheme in the real world. We finally conclude the paper in Section 8.

II. RELATED WORK

In this section, we review related work on secure data access control in the Cloud.

Yang et al. [28] introduced an extended proxy-assisted approach which weakens the trust required of the cloud server. Based on an all-or-nothing principle, the Cloud server’s private key is required for Cloud-side partial decryption. Consequently, in order for the cloud server to collude with another consumer to disclose a user’s cloud-side key, the cloud server would also have to reveal its own private key in order to perform a partial decryption. In our work, we address this collusion problem and prevent a revoked consumer from revealing the fully decrypted data even in the case of collusion.

Chen et al. [30] proposed the DataSafe architecture which involves bundling data with an access control policy and distributing the bundle to authorised users and applications. The policies are converted into hardware tags with parts of data associated with it, such as a portion of a document, electronic health records, etc and helps provide better access control where certain operations such as printing, storing to disk, viewing on monitor can be controlled. Even though the scheme doesn’t require that applications themselves be trusted, the scheme only applies to DataSafe compliant machines which contain special hardware, limiting the ability for user to gain access anywhere anytime. In our work, we similarly bundle encrypted data and keys inside a data object called SafeProtect. Our scheme is generic and not tied to a specific platform or application.

Squicciarini et al. [29] introduced 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 similarly to Chen et al. [30]. Relevant permissions are also created with the SCO. SCO networks (SCON) manage SCOs and also update each replica SCO with the latest change to address the Cloud storing multiple copies of data. The solution uses CP-ABE [31] 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. Furthermore, in our solution we make use of hardware-based TPM mechanisms which provides better resilience compared to software-based solutions.

Kayem [32] provided a solution which prevented authorised users from unauthorised data exchange. The solution uses an invisible digital watermark which is a hash of the encrypted data and key. Kirkpatrick et al. [33] described a solution which enabled data access only to known, tursted devices. A unique device is characterised by Physically Unclonable Functions (PUFs). This provides the best method of preventing data leakage and also the best level of authorised data sharing since the data owner can easily prevent unauthorised redistribution simply by checking if the device is recognised. 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.

Zic et al. [25] and Nepal et al. [26] proposed a technology called the Trusted Extension Device (TED), a TPM-based cryptographic controller, to enable mobile and portable trust in distributed systems. The TED plugs into any untrusted machine via USB and enables secure transactions with an institution. One of the benefits of this solution is that cryptographic keys are encapsulated within the device and is never exposed. All cryptographic operations are done inside TED via API calls. Since it is a hardware-based TPM, this makes it extremely difficult to retrieve the keys compared to software implementations which can be bypassed. We make use of TED in our work to demonstrate the concept of secure data sharing and access control in the Cloud environment.

In our previous work [34], we proposed a security model and protocol that enable secure data sharing in a remote telecare environment. A patient connects sensors to their body to measure ECG and runs an app on their smartphone. The app connects to the sensors via Bluetooth and reads the ECG data. The app then encrypts the data and sends it to the Cloud for storage. Authorised doctors obtain a key partition used to decrypt data while the Cloud has the other key partitions. The Cloud service partially decrypts the data using the key partitions and sends it to the doctor who then finally uses his key partition to fully decrypt the data. This solution is effective for data sharing since user revocation simply involves removing key partitions from the Cloud. However, it does not properly handle collusion attacks between user and Cloud since the Cloud can still keep copies of the removed key partition. In our work, we continue to use the idea of key partitioning and address the collusion problem.

Other solutions have studied data access control in distributed environments such as the Cloud but do not provide full data access control to the data owner and primarily focuses on protecting data from the Cloud. Goyal et al. [36] introduced fine-grained Attribute-Based Encryption (ABE) for access control of encrypted data. A user can decrypt data if his attributes satisfy the access policy of the encrypted data. Boldyreva et al. [37] suggested Identity-Based Encryption (IBE) where a user’s ID is used to generate keys to access data. Sahai and Seyalioglu [38] introduced worry-free encryption where a user can send files to others without worrying whether they have the right to access data. The solution uses functional encryption with public keys. In all these solutions, although the Cloud is assumed to be untrusted, the authorised user is always assumed to conform to operations allowed by the data owner. However, once the data is decrypted, the data owner loses control of their data and the decryptor can do whatever he wishes with it without being caught. In these solutions, the data consumer is assumed to be fully trusted. In our solution, we attempt to limit the trust barrier from the data consumer and assume the authorised data consumer to be curious.

In this paper, we combine our previous works [39] [40] [35] to provide stronger privacy and security when sharing data in the Cloud. [39] uses a hardware-based TPM module called the Trust Extension Device (TED) to enable secure sharing of data. The device attests with a certifying authority that it is legal and valid and decrypts the data if so. [40] bundles data and policy in a software object called SelfProtect Object (SPO). The policy is represented in the XACML policy language. The SPO evaluates the policy using environment attributes. If the policy is adhered to, the data would be released to the calling application. [35] proposes a software object which also similarly bundles data and policy, however, it also contains a background monitoring process which monitors the actions carried out by the data consumer on the decrypted data. Actions, such as copying or printing, are logged and periodically flushed to Cloud storage and notifies the data owner of any unauthorised operation.

III. PRELIMINARIES

A. Trust Extension Device

Zic et al. [25] and Nepal et al. [26] proposed a hardware-based TPM module called the Trust Extension Device (TED). This enables secure transactions to occur with an institution using an untrusted machine. We leverage the use of TED in our work to demonstrate secure data sharing and access in the Cloud environment since it can help prevent dishonest authorised consumers from illegally redistributing sensitive data to other consumers who do not have the relevant permissions.

Fig. 1. TED
Also, since TED is a hardware-based security mechanism, it will provide a much stronger protection compared to software implementations which can be easily bypassed. There are three components to the TED enterprise architecture [25]:

- **TED Issuer and Manager**: that is responsible for generating digital keys, issuing and revoking the TED as well as possibly responsible for the device’s manufacture.

- **A Privacy Certifying Authority**: that is responsible for verifying that the TED is valid and authentic. We use the trusted PCA in our work to authenticate and validate all data consumers personal TED devices. The trusted PCA can be modelled as a private Cloud owned by a small organisation.

- **An Application Server**: deployed within the enterprise, to perform the basic transactions required from the customer.

Since the TED contains its own VM OS and software, TED can also run in-built applications developed during the manufacturing of the TED device. In our work, we used TED to deploy our own application which will carry out cryptographic operations of symmetric keys as explained in our protocol in the next section. The cryptographic operations will be carried out without the host OS knowledge.

**B. ElGamal Encryption**

ElGamal encryption, invented by T. ElGamal is a public-key cryptography system [27]. We leverage the existing ElGamal Encryption in our work to strengthen the privacy of our system. We chose ElGamal encryption since the algorithm allows partial encryption/decryption. It is also simple and efficient. We incorporate a modified version of XACML policy language in our work. XACML is a generic, open-source access control policy language built on XML. We chose XACML for our policy since it can specify complex conditions upon which the data contents may be accessed and helps exemplify the idea that our solution can take a standard policy language and provide a complex and wide range of restrictions on the data limited only by the data owner’s requirements. The policy language allows the data owner to develop a policy that allows the data content to be accessed in a specific country for example, or within a specific time range within a day. The data owner can also enforce application-level policy control such as prevent a data consumer from copy/pasting their work or Save As a document.

We build a GUI-based tool that allows the data owner to freely specify a wide range of conditions over their data. The application takes the DOs inputs and generates a modified XACML document which will then be bundled along with the data as a SafeProtect object when it is time for storage.

Here is a sample list of conditions that a DO can specify on a document for example:

- no-copy: prevent the data from being copied.
- no-read: prevents the data from being accessed.
- no-print: prevents the data from being printed on a physical paper.
- play-times: the number of times the data can be accessed.
- location: the location where the data can be accessed.

Considering the scenario, we develop an example XACML policy for a data consumer 'Bob'.

```xml
<Subjects>
  <Subject>
    <AttributeValue DataType="...#string">Bob</AttributeValue>
  </Subject>
</Subjects>
<Resource>
  <AttributeValue DataType="...#string">testWord.docx</AttributeValue>
</Resource>
<Action>
  <AttributeValue DataType="...#string">read</AttributeValue>
  <AttributeMatch>
    <Condition FunctionId="...">
      <EnvAttrDesignator AttributeId="no-copy" DataType="...#boolean"/>
      <AttributeValue DataType="...#boolean">true</AttributeValue>
    </Condition>
  </AttributeMatch>
  <AttributeValue DataType="...#integer">5</AttributeValue>
</Action>
```

The policy states that data consumer 'Bob' can access the document 'testWord.docx' for a maximum of 5 times but cannot make a copy of the document at any stage.

**IV. SECURITY REQUIREMENTS**

From our scenario, we derive the following security requirements:

- no-copy: prevent the data from being copied.
- no-read: prevents the data from being accessed.
- no-print: prevents the data from being printed on a physical paper.
- play-times: the number of times the data can be accessed.
- location: the location where the data can be accessed.
Key Management: The data owner should be allowed to share data with a very large amount of users while also being able to efficiently revoke users without needing the re-encrypt and re-distribute encryption keys. Considering our scenario, Bob should be able to share health data with a group of doctors regardless of how large the group is and he should be able to revoke access from certain doctors.

Policy-based Control and Monitoring: The data owner should have greater control over his data no matter who is accessing the data from whichever location. The data owner should have greater trust that their data will not be misused by data consumers such as unauthorised copying or printing for instance. In our scenario, Bob should be able to specify that Dr. Alice cannot make a copy of the results and expect that the requirement will be enforced.

Authorisation: Data owners should be provided a stronger guarantee that the data is being accessed by the authorised data consumer only. No user should be able to forge credentials in order to access the data. In this instance, Bob shall be able to specify that only doctors in his personal group can access the test results.

Update Secrecy: The data owner should be able to update the data without having to re-distribute updated keys to all the consumers who have access to the data. That is, Bob should be able to update the health data without having to create another document and sharing the new document with his group of doctors.

Soundness: Any unauthorised user without authorisation should not be able to access the data provided by data owners. In other words, all other doctors and users who are not in Bob’s group should not be able to access the health data.

Auditing/Accountability: The data owner should be able to hold accountable any data consumer if they attempt to carry out any operation on the data that is not permitted by the data owner. In this case, Bob should be able to hold Dr. Alice accountable if she attempted to send his health data to another doctor without his permission.

V. Our Proposed Scheme

In this section, we first describe the main technologies that we used, then discuss our motivation behind our scheme followed by our models, assumptions and finally our protocol.

A. Technologies

1) SafeProtect

We leverage our previous work on SafeProtect [40] and use it in this work to enhance the security of data contents. Each SafeProtect object bundles the data owner’s data contents along with the policy. Figure 3 illustrates a simplified version of our XACML-based architecture. The API is used by the application requesting the data and the entity adding data to the object. After a data access request is made, the Policy Decision Point (PDP) interprets the policy and takes into account the environment settings by the operating user. The PDP then makes a decision whether data access should be enabled for the operating user. The decision response is sent to the Policy Enforcement Point (PEP). The PEP then obtains the data contents and sends this to the calling application via the API along with a set of rules on what operations are allowed on the data (e.g., no save, no print). The data contents in SafeProtect will not be released to the operating user until the policy has been interpreted and enforced. For more details on how this is done, please refer to our previous work [40]. It is then up to the application to carry out the policy enforcement. We assume the application to be trusted and do not focus on how the application enforces the policy. We have left this as future work.
2) Background Monitoring

We also build upon another of our previous works where we introduced the concept of background monitoring [35]. The background monitoring process aims to prevent a curious DC from carrying out unauthorised operations on the DO's data. Unless specified by the DO in the policy, the background process monitors whether the DC carries out any file operations such as save, copy, and print. If the policy states that the data should not be copied, the background process checks whether the DC carries out a copying operation on the data and if so, sends a log to the Monitoring Service and attempt to prevent the copying by deleting the data immediately. The Monitoring Service can then notify the DO directly that an unauthorised operation has occurred.

We modify the background process in this work by running the process from within the SaaS application. When the DC carries out an operation on the data within the SaaS application such as copying (using commands such as Ctrl-C for example), the process logs the action and sends to the PCA and subsequently the Monitoring Service. The DO can then retrieve through the Monitoring Service whether a DC carried out an action on the data and if so, sends a log to the Monitoring Service and attempt to prevent the copying by deleting the data immediately. The Monitoring Service can then notify the DO directly that an unauthorised operation has occurred.

We modify the background process in this work by running the process from within the SaaS application. When the DC carries out an operation on the data within the SaaS application such as copying (using commands such as Ctrl-C for example), the process logs the action and sends to the PCA and subsequently the Monitoring Service. The DO can then retrieve through the Monitoring Service whether a DC carried out an action on the data. The monitoring/logging mechanism is used mainly for auditing/accountability purposes. If the policy states that no copying is allowed, the SaaS application will disable all copy-based functionality from within the application and no logging would be required.

B. Models and Assumptions

1) Model

Based on our architecture model shown in Figure 2, here is a brief description of the functionality and services of each entity in our scheme:

DO: The owner of the data. The DO has the ability to store data to the DSS and keys to the PCA. The DO also has the ability to selectively share the data with certain DCs, and also revoke them on-the-fly by communicating with the DSS and PCA. The DO must also specify a policy (written in a simpler form of XACML) before storing the data in the DSS. The policy aims to provide restrictions on certain user types (e.g., preventing copy and save for student type users).

DC: The consumer of the data. The DC requests data access to the DO and if approved, has the ability to access the DO's data anywhere and anytime.

TED: All users, including DOs and DCs must obtain a physically-based TPM device called the Trusted Extension Device (TED). This is used by DOs to attest and authenticate the DO, as well as store/retrieve data to/from the DSS and provide data access approvals/revocations for DCs. The services of TED include:

- storeData: takes as input the data, DO public and private keys and outputs to DSS/PCA attestation certificates, and encrypted data and key contents.
- revokeUserAccess: takes as input the DCs email, and calls the DSS web service to revoke the DC by supplying email and attestation details.
- getFileList: call the DSS to obtain all accessible files for the calling DO/DC.
- getFile: takes as input the filename and a secret value, calls the DSS to obtain the encrypted data contents, decrypt the data contents and policy, and return to calling application.

Application: A Trusted SaaS application running within the PCA (similar to Google Docs). The application abides by and enforces the policy set out by the DO. A GUI Interface
is provided for both the DO and DC to carry out specific operations on the data. The DO uses the application to create their data and then store the data to the DSS using TED. The DC uses the application to access the DOs data using TED. Some of the services the application provides are:

- **storeData**: takes the data contents created in the application, and sends it to TED which later stores to the DSS.
- **getFileSizeList**: calls the TED to retrieve the file list for the DC and finally displays the files in a folder like structure inside the application.
- **getFile**: calls TED to retrieve the file and policy from DSS. Once file is received, the policy is evaluated, and the application modifies certain operations of the application according to the policy (e.g. disabling the save button).
- **logAction**: logs an action carried out by the DC on the application (e.g. copy/paste, print, etc). The action will then be sent to the Monitoring Service.

**DSS**: A Cloud-based storage service used to store DOs data in encrypted form and provide access to DCs via TED. Some of the services of DSS include:

- **storeData**: stores the supplied data contents and encrypted key to Cloud storage.
- **removeData**: removes the supplied data contents and keys from Cloud storage provided a data id is supplied.
- **getFileSizeList**: retrieves a list of filenames that the DC is allowed access to using the supplied DC credential information.
- **getData**: retrieve the data stored in the Cloud, provided the data id is supplied.

**PCA**: A Trusted Cloud-based service used to attest that the DO/DC is really who they say they are via TED. The PCA also stores DC keys and DO data keys. The PCA can also be used to revoke certain DCs from access to the data. Some of the services of the PCA include:

- **authenticateUser**: Used to authenticate calling DO/DC using the attestation details provided by their corresponding TED.
- **storeDataKey**: stores a key partition needed for data access of the DOs data.
- **deleteDataKey**: deletes a key partition needed for data access of the DOs data.
- **addUser**: adds a DC to the list of DCs who have access to the DOs data.
- **revokeUser**: revokes a DC from list of DCs who have access to the DOs data.

**Monitoring Service**: A Cloud-based storage service that stores application-based actions carried out by DCs. DOs can use this information for auditing/accountability purposes. The Monitoring Service mainly interfaces with the SaaS application and PCA. The main services include:

- **store**: stores the action as an entry in the database containing the DC identifier, data identifier and the action itself.
- **get**: retrieves the action carried out by the DC on the data.

**SafeProtect**: A software-based object containing data contents and XACML policy. The object contains a PDP which interprets the policy and environment and indicates whether data access should go ahead. The PEP is used to enforce the policy. SafeProtect objects remain in TED and releases the data contents to the host machine and application only when the calling user and environment satisfies the policy. The API is the main source of communication with the object. Some of the services include:

- **getData**: obtain full data by supplying attributes. These attributes are used by the PDP to determine whether data access should go ahead to the calling user. SafeProtect will evaluate the policy using the supplied attributes. For more details on how this is done, refer to our previous work [40].
- **updatePolicy**: The DO can update the policy bundled in the object. No other user will be able to carry out this service.
- **storeData**: The DO stores data contents in the object along with the policy file.
- **generateSafeProtect**: Generates a new SafeProtect object and stores data contents along with the policy file. Only the DO will be able to call this service.

2) **Assumptions**

Throughout the paper, we assume that the DO and DC registers an account with the DSS/PCA and provides all necessary verification details which proves the user is who they say they are. TED devices will not be shipped to users who fail this verification stage. We also assume that DC’s are aware of the terms and conditions of accessing SafeProtect data. For instance, one of the terms can state “The Data owner has the sole right to define the policy over his data in any way he wishes. Any attempts to go beyond the rights set out by the data owner will be considered unauthorised as per the terms and conditions”. If the DC performs operations beyond the policy, it would be treated as unauthorised and the responsibility will be left upon external law enforcements to serve justice to the DC. Hence DCs should abide by terms and conditions. Furthermore, the policy is reflected and enforced within the application and hence DCs should be well aware and abide by the rules.

The policy can only be enforced based on applications running in the Cloud (i.e., SaaS based applications). Applications on a consumer’s personal computer is always under the full control of the consumer since the consumer can do anything he wants with the applications under his operating system. For example, he can write a new plugin to print the document which can bypass the policy enforcement. Thus, the policy can never really be enforced on consumer applications. SaaS based applications provides the DO the ability to enforce the policy to a greater degree.

We also do not focus on other types of attacks such as taking a photo/screenshot, memory-based attacks or manually copying the data as this is beyond the scope of this paper. We do however, assume the DCs are made aware of the terms and conditions before they access the data which states that the DO has rights to define the policy on the data and any attempts to perform operations beyond the policy will be classified as unauthorised. The Trusted application will help to enforce the
Our scheme consists of six phases, namely, initialisation phase, data owner storage and set up phase, data share phase, file access phase, data access phase and data consumer revocation phase. In this paper, we closely focus on the data owner storage and set up phase, the data share phase and the data access phase as they are the most relevant to this paper. The other phases of the protocol can be referred to in detail in our previous paper in [39]. We will use the notations from the previous sections to describe our protocol. The formal descriptions for each of the protocol phases are provided in figures 4, 5 and 6.

1) Owner Storage and Setup Phase

In the owner storage and set up phase (shown in Figure 4), the DO securely stores his data in the DSS and configures the data for data sharing. The DO sends the data and policy to TED. TED will create encryption keys and store the encrypted data contents and policy to a SafeProtect object. TED then sends these details to the DSS and also sends a key partition (which sets up the data sharing) to the PCA. We describe this in detail in the following steps:

We first consider data protection:

O1. The DO writes his document and creates a policy for how the document should be used.
O2. The DO connects his TED and sends the data and policy to the device.
O3. TED will then generate an arbitrary public and private ElGamal key pair for the data.
O4. TED then partitions the private ElGamal key into 2 pieces.
O5. TED will then generate an arbitrary symmetric key $k$.
O6. TED will then symmetrically encrypt the data using $k$.
O7. The symmetric key $k$ will then be encrypted using the public ElGamal data key and then partially decrypted ($P_dK$) using the supplied key partition.
O8. TED will then generate a SafeProtect object SO which bundles the encrypted data and policy and send it to the DSS, along with credentials and signatures per TED architecture.
O9. The DSS requests the PCA to verify the signature of the TED signed endorsement credentials and the DO credentials.
O10. The PCA, after successfully verifying the signatures and credentials, returns the DO id to the DSS.
O11. The DSS stores the SafeProtect object, partially decrypted symmetric key and DO id in Cloud storage and returns the file id back to TED and similarly to the DO.
O12. TED then requests the PCA to store the remaining ElGamal key partition along with the file id.
O13. The PCA verifies the credentials and signature and obtains the DO id.
O14. The PCA stores the DO id, file id and remaining ElGamal data private key partition to its database.

2) Data Share Phase

In the data share phase (illustrated in Figure 5), the DC requests access to the DO’s data and registers to the PCA by making a payment. When the payment is approved, the PCA further partitions the supplied key partition from the previous phase and creates a record for the DC containing one of the generated partitions. The other partition is sent to TED. We describe this in detail in the following steps:

S1. The DC sends a share request to TED with the file id and that share request is forwarded by TED to the PCA.
S2. The PCA verifies the signature and credentials of the DC.
S3. Once successfully verified, the PCA verifies the file id exists in the database.
S4. If the file id exists, the PCA makes a payment request directly to the DC.
S5. The DC responds with payment information.
S6. The PCA processes the payment.
S7. If successful, the PCA uses the key partition supplied by the DO and partitions it further into 2 parts.
S8. One partition is stored in the database pairing along the DC id and file id.

S9. The other partition is encrypted with the public ElGamal key of the DC.
S10. The PCA sends the encrypted key partition and file id to the TED.
S11. TED decrypts the key partition using the encapsulated DC private ElGamal key.
S12. The other key partition is then stored along with the file id in TED storage. Note, this key partition value will never be known to anyone other than TED.

3) Data Access Phase

In the data access phase (highlighted in Figure 6), a DC accesses the DO’s data. The DC first sends a request to access the data using TED. TED will then send a request to the SaaS App. The retrieves the SafeProtect object and key partition from the DSS after attestation with the PCA. The App calls the SafeProtect object to retrieve the data and if the object’s policy is abided by, it will return the encrypted data contents. The app joins the remaining key partitions to decrypts the symmetric key and subsequently, the data. Once the data is displayed through the App, the background monitoring operation begins and logs every operation of the DC. These logs are periodically flushed to the Monitoring Service. Below are the specific details of this phase:

A1. The DC first connects the TED and requests access to the data by supplying the file id and credentials.
A2. Once connected, the TED sends a file access request to the Cloud-based Application running in the PCA and sends along credentials, signature and file id.
A3. The Application the requests the DSS to obtain the SafeProtect object.
A4. The DSS requests the PCA to obtain the DC’s partial key associated with the file id.

A5. The PCA returns the DC’s corresponding partial key after verifying credentials and signature.

A6. The DSS obtains the SO from its Cloud database using the file id.

A7. The DSS also obtains the partially decrypted symmetric key from its database (PdK) and further partially decrypts with the key supplied by the PCA.

A8. The DSS sends the SO and the partially decrypted symmetric key to the Application.

A9. The Application sends a request to the PCA to get the remaining partial key for the DC.

A10. The PCA calculates the remaining key partition after subtracting the DO key partition with the DC key partition, and sends the result back to the Application.

A11. The Application fully decrypts the PdK using the key partition supplied by the PCA. Note, if the subtraction operation in the previous step failed (i.e., the key partition to the data is not valid), the symmetric key will not be fully decrypted, since priv(data) = x1 + x2 = x1 + x3 + x4.

A12. The Applications obtains the data contents from the SafeProtect object. After the SafeProtect object evaluates the policy and the environment, it releases the data under a set of policy conditions to the Application.

A13. The Application then uses the fully decrypted symmetric key to decrypt the data contents.

A14. The Application enforces the policy and disables commands (e.g., save, save-as, copy, paste) if specified in the policy.

A15. The Application then displays the data to the DC.

A16. The DC then carries out an action on the Application such as copy/paste, save, print, etc.

A17. Each action by the DC is logged by the Application and sent to the PCA along with the credentials and data identifier.

A18. The PCA sends the action along with the identifiers to the Monitoring Service.

VI. SECURITY ANALYSIS

We now provide an informal security analysis of our protocol.

- Data Monitoring - The policy file bundled inside SafeProtect uses a modified version of XACML policy language. This allows complex policies to be created. If the conditions fail to meet the policy, SafeProtect will output 'Deny' and the data will not be released. For instance, we can prevent DCs from making a copy of the DO’s data and sending it to their friends. As long as the DO
states that the data shall not be distributed in the policy, the SaaS-based Application will disable all controls such as save/save as, copy/paste and print commands from the online application, thus preventing the DC from being able to send the data over to their peers. Furthermore, the DO can specify that background monitoring should occur to monitor the actions of the DC on the data. The monitoring process is built from within the SaaS application and thus it is difficult for an outsider to break or prevent the monitoring process from running. It is impossible to completely prevent a dishonest DC from redistributing the data, since the DC can simply take a photo or screenshot of the data and send it to their peers or even physically handing over the TED to their peers. However, this is beyond the scope of this paper.

**User Revocation** - In our protocol, user revocation can be achieved efficiently without the need to re-encrypt the data each time. The DC’s key partition as well as credential information associated with the data is simply removed from the trusted PCA’s database. This way, if the revoked DC attempts to access the data, their corresponding TED will fail to be attested and furthermore, will never be able to be decrypted without the key partition.

**Collusion between DC and DSS** - In the event a dishonest DC and the untrusted DSS collude, the DC will only be able to retrieve the encrypted data and the partially decrypted symmetric key. Without the remaining key partition, it is extremely difficult for the DC to decrypt the data as the data decryption key will still remain illegible. Since the remaining key partition is encapsulated within the TED device, the DC would have to carry out hardware-based attacks in order to retrieve the remaining key partition. Hardware-based security mechanisms are much more difficult to attack and hence our protocol handles collusion attacks between DC and DSS.

**Man in the Middle Attacks** - Our protocol handles man-in-the-middle attacks since everytime a DC requests access to the data, the TED application always attest with the DSS and PCA using the randomly generated attestation identity key. The encrypted data and keys are returned to the TED device with a signature by the PCA attesting that the TED device is legal and valid.

**Update Secrecy** - When the DO wishes to replace his current stored data with a newer updated version, he simply encrypts the data with the same symmetric key and sends it to the DSS to be replaced. Note that nothing else in the protocol needs to be changed, hence making our solution practical to be deployed in the real-world scenario.

**Losing TED** - In the event that a DC’s TED device is physically lost or stolen, the DC immediately notifies the TTI (using his credentials) that the device is stolen. The TTI verifies the credentials and notifies the PCA to remove the entry containing the DC’s credentials and key partition.

**Insider Attacks** - Our protocol is also secure under insider attacks since there is never a stage in our protocol where the data is decrypted in the Cloud. The data remains encrypted at all times on the untrusted Cloud servers as well as on untrusted public communication channels.

**Auditing/Accountability** - Our protocol also enables auditing and accountability through the monitoring mechanism. Every action carried out by the DC on the SaaS application is logged by the application and sent to the Monitoring Service where a DO can then retrieve actions and hold accountable if necessary.

VII. IMPLEMENTATION AND EVALUATION

In this section, we describe the implementation details of our system, followed by performance tests and evaluation.

**A. Implementation**

To demonstrate our ideas, we built a plugin for Microsoft Word. To develop our plugin we used Visual Basic .NET to develop a Word solution. The data model of our system is shown in Figure 6. A Word document contains a set of properties. These properties can include author name, year, date modified, etc. Microsoft Word also allows users to define their own custom properties. We took advantage of this fact and bundled a text version of our XACML policy as a property. For added security, the policy was encrypted and the ciphertext was set as the property. This property acted as our PDP. Our plugin itself acted as the PEP since it would retrieve and decrypt the encrypted policy as soon as the document was loaded, and would parse and interpret the policy. The plugin would then inform the application to disable certain functionality such as Save/Save-As or Print.

Figure 7 illustrates our plugin disabling save and print operations to prevent data consumers from making an extra copy of the data. The interface allows the data owner to add an encryption key used to protect the document as well as set a number of permissions that should be restricted by data consumers. For instance, if the data owner requires that his data consumers must not take a physical printout of his document, he simply toggles the ‘No Print’ checkbox. The plugin will generate the XACML file as soon as the data owner presses the ‘Send to the Cloud’ button. Not that the data contents become encrypted as soon as the data owner presses the ‘Send To SafeProtect’ button. Figure 8 illustrates the interface that will be seen by data consumers when they access the data owners data. As long as they data consumer has the corresponding key, they will see the fully decrypted document. The plugin will also interpret the XACML policy and enforce it within the application. Going by the previous example, if the data owner specifies ‘No Print’, then the application will disable all print options. Similarly, the same would occur for ‘Save’, ‘Copy’ and ‘Modify’ operations. In Figure 9, the application disables the save and print operations based on the policy created by the data owner.

We used a temporary folder to represent TED. The plugin would store the protected document containing the data owner’s data as well as policy in the TED folder. When a data consumer requests access to the document, the plugin opens the same document from the TED folder and carries out decryption as well as policy enforcement.
B. Performance Tests

We carried out a number of tests to measure the performance of our prototype system. We measured the performance of securely storing as well as accessing the data owner's document. The main purpose of these tests was to validate the feasibility of this system in a real-world scenario. We ran 10 test cases to measure first the secure uploading stage and then later, the data access stage. The tests were carried out on an Intel Core i5 HP Pavilion Notebook with 4GB memory, 1TB storage and 64-bit Windows 8 operating system.

We first carried out tests on multiple test documents containing 1 page, 2 pages, 4 pages, 10 pages and 20 pages. Table I provides details of each of the test documents we used during the performance tests. We noted down the time it took to encrypt the data as well as policy and finally send to the TED directory. Figure 10 highlights the results of our performance test.

<table>
<thead>
<tr>
<th>Document No.</th>
<th>Pages</th>
<th>Size</th>
<th>Words</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>17 KB</td>
<td>433</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>29 KB</td>
<td>853</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>48 KB</td>
<td>1721</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>114 KB</td>
<td>2994</td>
</tr>
<tr>
<td>5</td>
<td>20</td>
<td>230 KB</td>
<td>12674</td>
</tr>
</tbody>
</table>

**TABLE I: Test Documents**

![Fig. 7. Send document to Cloud](image1)

This is a test document.

![Fig. 8. Open document from Cloud](image2)

![Fig. 9. Policy Enforcement](image3)

![Fig. 10. Data Protect and Send Performance](image4)

We then carried out tests with the same test documents to measure the time it took to obtain the document from the TED directory, decrypt and enforce the policy, and finally decrypt the document. The results are illustrated in Figure 11.

![Fig. 11. Data Access Performance](image5)

C. Evaluation

From the data protect and send performance tests, we found that there was only a slight increase in time when the more pages of text was added. However, even for documents as large as 20 pages, the protection algorithm still took significantly
less than a tenth of a second. When the number of pages doubled, the performance times only increased slightly, hence our protocol would not be expected to create as much overhead compared to simply opening a normal Word document.

Regarding the data access tests, performance times were considerably larger, with the 20 page document access taking nearly a quarter of a second to access. We also found that doubling the pages, nearly doubled the performance times. This is clearly visible in the performance times with the 10 page document, compared to the 20 page document. However, even with a document as large as 20 pages, the data access times were still feasible since access times were well under one second. Hence our current prototype system, would be feasible to implement in a real-world scenario.

D. Usability Evaluation

Though we have not performed any formal usability evaluation for the SaaS-based application, we make the following observations about the usability of the system. The system has to be evaluated in terms of a) Ease of Use and b) Trust and Security.

- Ease-of-Use - Since we used Microsoft Word to represent our SaaS-based application, the ease-of-use of our application is the same as the ease-of-use of Microsoft Word. Thus, we do not need to carry out any ease-of-use usability study for this work.
- Trust and Security - We used the same security protocol from our previous health application in this work. Thus, the trust and security evaluation in this work would be the same from our previous work. We originally carried out the usability study on 10 potential participant users and they were asked questions on how they felt about the security of our system. We provide the chart on the participant responses below.

![Usability Chart]

From our trust and security results, more than 70% of participants had at least some concern to what others see when they exposed data over the Internet. Over 50% of participants felt that their data would be kept private and secure when using our protocol and were made more aware of the type of information that they may be exposing over the Internet. Nearly half the participants agreed in regards to whether their personal information will be managed carefully and not leaked to the outside.

VIII. Conclusion

In this paper, we have proposed a model and protocol that will enable private and secure data sharing in the Cloud that provide data owners greater control and ownership over their data using data owner defined policies. We achieve this by combining a software-based data object called SafeProtect and a hardware-based TPM device called TED that help to enforce the policy set out by the data owner. Although we used Microsoft Word for our prototype, we argue that the idea can be incorporated into other popular applications used for data access.

As part of future work, we aim to further improve our security model and protocol. We will be looking to further improve trust in data owners by Requesting proof by applications that the policy is enforced. For instance, in the event that the data owner requests no printing of their data, SafeProtect can request access to the printing port to see if the data is being printed. We are also looking to prevent screenshot and memory-based attacks to further enhance policy enforcement.

REFERENCES

Danan Thilakanathan Recently graduated from The University of Sydney in Bachelor of Engineering (Software) with Honours and is doing his Ph.D. in collaboration with the CSIRO and The University of Sydney working on privacy and security in Cloud technologies with a focus on data sharing and collaboration.

Shiping Chen IT professional with over 20 years research experiences and combined R&D skills. Since joining in CSIRO in 1999, he has worked on a number of middleware-related research and consultant projects, including software architecture, software testing, software performance modelling and trust computing. He has published extensively in these areas, ranging from academic research papers to in-depth industry reports. In the past several years, he has been working closely with the University of Sydney by co-supervising Ph.D./Master students and actively involved in research community services on web and service computing areas. His current research interests include service computing and cloud computing.

Surya Nepal Principal Research Scientist working on Service and Cloud Computing. He has also worked on trust and security aspects of collaboration at CSIRO ICT Centre. His main research interest is in the development and implementation of technologies in the area of Service Oriented Architectures (SOA) and Web Services. He completed his Ph.D. with RMIT University, Australia in 2000, and then joined the CSIRO Division of Mathematical and Information Sciences. Surya undertook research in the area of multimedia databases, web services and service oriented architectures, and security and trust in collaborative environment. He has several journal and conference papers in these areas.

Rafael Calvo Professor at the University of Sydney. He has a Ph.D. in Artificial Intelligence applied to automatic document classification. He has taught at several Universities, high schools and professional training institutions. Rafael also has worked as an Internet consultant for projects in Australia, Brazil, the US and Argentina. Rafael is the recipient of 5 teaching awards, and the author of two books and many publications in the fields of learning technologies, affective computing and computational intelligence. Rafael is Associate Editor of the IEEE Transactions on Learning Technologies and of IEEE Transactions on Affective Computing and Senior Member of IEEE.