Portable Key Management Service for Cloud Storage

Julian Jang-Jaccard
ICT Center
CSIRO
Australia
julian.jang-jaccard@csiro.au

Avnish Manraj
ICT Center
CSIRO
Australia
avnish.manraj@csiro.au

Surya Nepal
ICT Center
CSIRO
Australia
surya.nepal@csiro.au

Abstract— **Cloud storage services provide highly scalable, available and pay-as-you-go storage space for individual and enterprise users. Cloud storage services are inherently insecure as the management of the data in the cloud storage is controlled by third parties beyond the reach of the data owner. To address this problem, a number of data obfuscation techniques have been proposed to conceal data before sending it to the cloud. The secrets keys used for obfuscation are stored in a secure location while obfuscated data is stored in the cloud. In these approaches, the data is as secure as its corresponding keys. However, this still brings a challenging issue where a user needs to manage a large number of (secret) keys in such a way that they are protected against all types of adversaries, and should be as highly available as cloud storage services. To address this issue, we propose a portable key management service that is highly secure and available. In our solution, all keys are stored in a tamper-proof hardware within a portable USB device that users can carry with them all the time in order to provide high security and availability. We describe the system model, the details of the key management service and a prototype implementation.**

Keywords- Cloud Computing, Key Management, Portability, Storage Service, Trusted Computing

I. INTRODUCTION

Moving data into the "Cloud" offers great cost saving and convenience to users and provides a new platform for collaboration. Users do not have to concern about the complexities of data storage capability in house. Using the increased network bandwidth and flexible network connection, users can simply subscribe a service on "pay-per-use" basis and access to huge storage space.

This new cloud storage paradigm has, however, brought many security concerns [8], [19], [20], [21]. One of the biggest concerns is protecting confidentiality and integrity of data. Under cloud storage paradigm, the data storage and management is under the full control of (untrusted) third parties (i.e., the cloud storage service providers). The data owners are left vulnerable having to solely rely on the security mechanisms and configurations of the cloud storage providers to protect their data [6], [9], [10].

Two most common security properties that would suffer as a result of using cloud storage services are data confidentiality and data integrity. Data confidentiality would be broken if user's data is disclosed to unauthorized individuals. For example, malicious hackers may penetrate the storage servers and steal the data using malware infection. Data integrity could be violated if user's data is modified by unauthorized individuals. For example, financially motivated personnel at the data centre may tamper private information without permission.

In addition, data exploitation does not end when the user's subscription of cloud storage services expires. The data owner often falls into a trap thinking that their data is safely removed from the cloud storage once their subscription is expired. This can be far from reality [18]. It is a common practice for a large data centre to make extra copies of the original data. For example backup media are used to store a copy of the data to use it to recover from unexpected disaster. In another example, replicated databases contain any copies of the data to guarantee high availability. The backup media or replicated databases with copies of the original data may or may not be completely removed when the user's subscription expires. It is more than a possible scenario that user's data still resides somewhere in the data center and is subject to misuse. Therefore, the data owners should always be vigilant in safeguarding their data all the time such as while data is at transfer, at rest, and at use during and beyond the subscription.

To address this problem, a number of solutions have been proposed in academic literature and industry products. The fundamental concept behind these solutions is to obfuscate user's data before sending it in such a way that it is not possible to reveal user's data to unauthorized third parties (i.e., cloud storage providers or adversaries) other than intended users (i.e., data owners). Fragmentation is often used in conjunction with obfuscation techniques [27], [28]. The original data is often fragmented in many smaller pieces. Each fragment is then encrypted using a unique key. The encrypted fragments are stored in the cloud while the unique keys are stored in a secure location away from prying eyes of third parties. Some of the popular choice of the secure location include such as a dedicated file system in the user's local machine [10] or in a private cloud [29].

However, this still brings a challenging issue where a user needs to manage a large number of (secret) keys. It would become increasingly troublesome for users having to manage hundreds of keys as a result of fragmenting a large size data, or dealing with multiple cloud storage providers (using different keys for different providers). Furthermore, the high availability of data provided by cloud storage services can be easily compromised if the keys are not

COLLABORATECOM 2012, October 14-17, Pittsburgh, United States
Copyright © 2012 ICST
DOI 10.4108/icst.collaboratecom.2012.250488

available all time. If that happens, it can significantly damage the reputation and success of the cloud storage solution. Henceforth, the keys should also be as highly available as data itself in order to achieve the high availability in overall secure cloud storage solution.

To address this issue, we propose a portable key management service that centralizes all the expense and expertise required to maintain a large number of keys. Our solution is still highly secure and available. All keys managed by our key management service are secure as the keys are stored in a designated sealed storage area within a tamper-proof hardware device. This ensures that the general stealth of unique keys, especially by the use of remote Internet connections which are on the rise, is much more difficult. Our key management service is highly available as they are not locked into any particular machine. Our key management service is implemented in a portable USB device that users can carry with them all the time and use it at any time as needed.

The rest of the paper is structured as follows. In Section II, we first demonstrate problems in existing system model, run threats analysis and define security requirements. Then, we provide a system model we propose. In Section III, we describe major design considerations that are important in developing a key management service which can protect secret keys, user's data and the platform it is running. In Section IV, the details of the key management service are described including key creation, distribution, aiding encryption, and key deletion. In Section V, we illustrate an attestation protocol that runs on the user's machine to ensure the platform configuration and operations match the expectation. In Section VI, we describe our prototype implementation and the key functionalities implemented at major components of our system. In Section VII, we present the related work. The last Section VIII presents the concluding remarks and future work.

II. PROBLEM STATEMENT

We first describe an existing system and then illustrate potential threats that may occur at various points while user's data traverse. We define a number of security requirements to mitigate the threats and propose a system model that accommodates them.

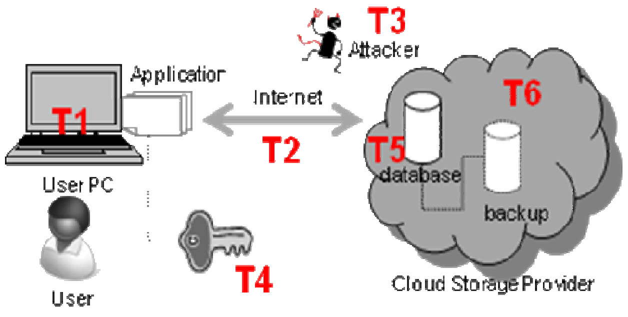

Figure 1. Potential Threats in Existing System

A. *Possible Threats*

Most cloud storage service today provides a web application where a user can upload (and subsequently download) data using their own PC at comfort of his/her home. Some cloud storage provider may support an extra service (e.g. via a third party plug-in) where user's data is processed using an obfuscation technique to provider more security before it's send to the data center. This existing model is depicted in Figure 1. However, many things can still go wrong to breach the data confidentiality and integrity. We illustrate a number of potential threats that could violate data confidentiality and integrity of user's data at various points on the pathway from user's PC to the cloud.

- *T1.Dishonest local host machine intercepts user's data.* Most naïve users do not have a skill set to protect their machines free from malware. Malware installed in the user's machine without the knowledge of the user may spoof user's activity or worse send the secret key information to the attacker.
- *T2. Man-in-the-middle intercepts user's data.* If user's data is improperly protected while in transit between the cloud user and the cloud storage provider, malicious man-in-the-middle may steal the data and make unauthorized access to the user's data.
- *T3. Malicious attacker pretends to be the cloud user*. Malicious attackers may pretend to be a legitimate cloud user and then send in a massive amount of data occupying a large data storage space and subsequently disrupts a proper operation of the data centre. If that happens, the cost to the end user in terms of dollars also could be very high, as the service works on pay-per-use basis.
- *T4.Malicious attacker pretends to be a key management service.* The key management service knows all secret keys that are used in the encryption process. Data leakage can happen if a cloud user mistakenly sends in plaintext or cipher to a rogue key management service.
- *T5. Dishonest cloud provider makes an unauthorized attempt to read user's data.* The cloud storage provider has all necessary tools and mechanisms to access the data that is under its full control. Either by a malicious code planted in the data center or by deliberate attempts by a dishonest data center personnel, data leakage can easily happen if user's data is improperly protected.
- *T6. Dishonest cloud provider makes copies of user's data.* It is a common practice for cloud storage providers to make copies of original data and store them in extra storages, such as in the backup media or replicated databases. Theoretically, these copied datasets are then re-applied if any unexpected disaster happens at the data center so that user's data can be recovered. Other cases, replication is used to guarantee the availability of data. However, even

after user’s subscription expired, these copies of the original data can still remain somewhere in the data center and may become a subject for compromise if the data is improperly protected.

B. Security Requirements

- R1. The encryption process happens at a sealed environment so that the host machine could not eavesdrop or steal any important details of both the key information and key computation (addressing threat T1).
- R2. The secret key used for the encryption is securely protected so no snooping or stealing of the key is possible (addressing threat T1 and T2).
- R3. Data encryption is done in a way that it does not leak its original content other than to authorized systems or individuals (addressing threat T1, T5 and T6).
- R4. Data integrity is always checked after receiving the data to ensure tampering has not occurred during the transmission (addressing threat T2, and T4) or at rest.
- R5. A cloud user is authenticated by the cloud provider so that the data storage space is utilized only by the authenticated users but no one else (addressing threat T3).

C. Our System Model

We proposed a new system model that we believe to be prepared better to rectify the potential threats. Our model is designed to accommodate all the security requirements we defined above. Figure 2 depicts an overview of our proposed portable key management service with the descriptions of its core components.

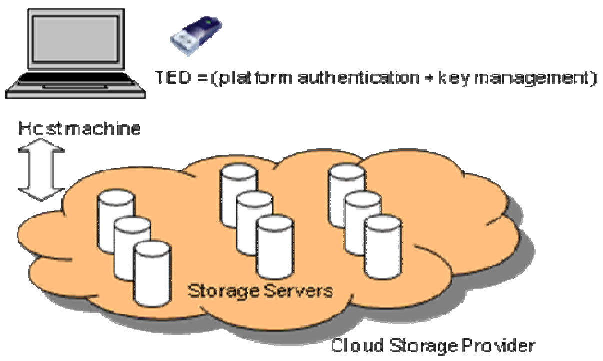

Figure 2. The overview of our proposed system

Trust Extension Device (TED): The Trusted Computing Group (TCG) [2] defines a set of specifications aiming to provide hardware-based root of trust and a set of primitive functions to propagate trust across remote platforms. The core of TCG specifications were realized and implemented based on Trusted Platform Module (TPM). TPM is a cryptographic microcontroller system, which was typically embedded on a motherboard in a PC. However, the PC-based TPM solutions have been criticized for its impracticability, especially with its portability issue that the TPM is locked to a machine. It is also criticized with the difficulty for doing sensible integrity measurement due to the size of machines today [25], [13].

Improving from existing shortfalls of PC-based TPM, we previously proposed a portable USB-based security device dubbed as Trust Extension Device (TED) [12], [30], [31]. The picture of TED is shown in Figure 3. TED is essentially considered to be a portable TPM chip which can be plugged into any host machine using a USB connection, still providing all necessary TPM functionalities.

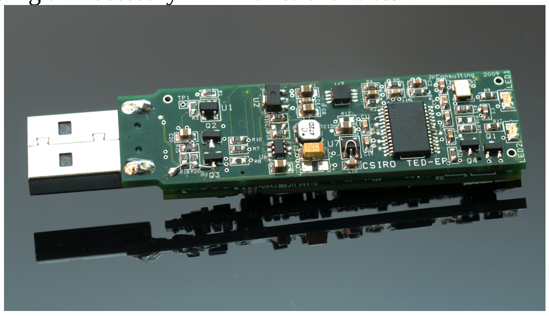

Figure 3. A picture of our TED

We have made further improvement to our existing TED to provide two additional mechanisms: high assurance platform authentication and key management service.

- High assurance platform authentication: it provides a mechanism where the data owner presents a piece of evidence (i.e. proof of compliance) to the cloud storage server. The proof contained in the evidence establishes two facts to the storage server: (1) the requests are indeed sent from the legitimate user, (2) and the requests are produced by the machine with configuration and operations free from running any malicious software.
- Key management service: it provides an expertise in maintaining a large set of keys and performs operations securely. The functions provided by the key management service includes key creation, key distribution, assisting in data encryption, providing secure data storage, and destroying the keys that are no longer needed. It also ensures that the keys are always available on demand.

Local Host: We assume that the environment where a local host runs is unknown and untrustworthy. The TED is equipped to create its own trusted environment when it is plugged into the local host. The local host acts only as a medium that connects and transmits the data between the TED and the remote server via the public Internet facility.

Cloud Storage Provider: The cloud storage provider maintains a large data center and provides a set of services to access and manage the data for the cloud users (i.e., data owner). The cloud user pays for a subscription to use a certain amount of storage space for a fixed amount of period. Though the cloud storage provider may have security mechanism in place to protect the data they maintain, the cloud user does not necessarily fully trust neither the provider themselves nor the security mechanisms supported by the cloud storage provider.

III. Design considerations

We describe major design considerations which we believe to be important in developing a key management service. This includes the considerations to protect secret keys, user's data and the platform it is running. We describe our approaches in providing solutions to each of the design consideration as follows.

A. Secret Keys

We describe a number of critical considerations that we feel important to address in managing a large number of secret keys. This includes who is responsible for managing the lifecycle of each key and where the keys are stored. We also describe the portability and the importance of the high availability of the keys.

1) Key Management Responsibility

One solution to protect data from untrusted cloud storage provider would be to encrypt user's data using a secret key before sending the data to the cloud. This provides two important advantages for the data owner. By encryption, the content of the user's data is never revealed to unauthorized third party. It is also somewhat easier to make the data unrecoverable by simply deleting the secret key. The question is then who is going to maintain the control of the secret key. It is especially so if there is a large number of keys created as a result of fragmenting a rather large sized data or if the user decides to store different data sets in the number of multiple storage providers (so that only the portion of data is lost if a certain storage provider experiences failure for any unforeseen reasons). It would be unrealistic to leave the responsibility to the users to manage such a large number of keys. For example, if Alice decides to fragment her data file into many smaller size pieces each of which is encrypted using a unique key, it would place a huge burden for Alice to create, reliably store, certify, advertise, and then reliably destroy potentially a large number of keys.

To overcome this problem, we adapted an approach based on R. Perlman's Ephemerizer [17]. The central idea behind the concept of the Ephemerizer is to have a single service which centralises all the key management expertise in one place. The service creates keys, makes them available for encryption, aids in decryption, and destroys the keys at the appropriate time. In the original paper, however, the Ephemerizer acts as a trusted third party service in public space whose responsibility is solely to manage keys for many different users. We modify the scope of the Ephemerizer. Our key management service does what original Ephemerizer does by concentrating all the expense and expertise in managing keys in one space. However, our key management service is implemented within a dedicated hardware device in our TED. This design ensures that the stealth of the keys is more difficult and key computation is secure from any remote attackers. In addition, the key management service is offered to only a single user (i.e., the TED owner). This design simplifies the complexity of having to deal with many users therefore potentially avoids the concurrency issue of dealing many keys for different users simultaneously.

2) Key Duration

Commonly, the cloud storage provider employs strategies to keep user's data in additional storage media, such as backup files or replicated databases, often without user's knowledge. If the keys that encrypt user's data are never destroyed, encrypted data is more vulnerable from the further misuse. For example, it is reported in [17] that the long term keys are made more easily available through compromise over time. In worst case, they become recoverable if a technique such as forensic is applied.

To avoid such possibility, we propose to create short-term keys which have expiration dates attached to them. This ensures that the keys are regularly monitored and deleted after their expiry dates. The data encrypted by the expired keys becomes unrecoverable by any means. For example, if Alice is to send an encrypted data blob to a cloud storage provider that will be removed after 3 months, then it is better to be encrypted in a key that will be guaranteed to be available for only 3 months but no more. By deleting the key after 3 months, the encrypted data with the expired date becomes irrecoverable by any means as the key is no longer available. Though it is not currently implemented, it would be straightforward to extend our current model for some users whose data resides in the data center permanently. For example, a special date format such as 99/99/9999 can be used for data such as that.

3) Key Storage

Another important question that needs to be addressed is the location of the keys. If the keys are located in an improperly protected place, they become vulnerable from malicious attackers. For example, with sophistication and skill sets, a hacker with deep knowledge in OS could easily steal the secret keys reside in a stable memory space. In another example, if direct stealing of the keys is not possible, the attackers could use spoofing malware to monitor the key computation and guess the keys. To ensure such compromises are not possible, we propose to implement the key management service inside our tamper-proof device TED. The TED provides a sealed storage area to store the keys. The public part of the key is used to assist the encryption while the private part of the key is securely held inside the TPM chip. The private part of the key is never exposed outside the device (*satisfies the security requirement 2*).

4) Key Portability and Availability

Inherently, a key management service is implemented at a server or at a private cloud. These keys are then locked into that particular server or the environment. If users want to use the key management service to encrypt a certain portion of their data, they need to return to the server machine to perform the intended operations which may not be always convenient.

This limitation is mitigated in our proposed solution as our key management service is implemented within a portable USB. This design greatly improves both the portability and the availability of the keys. Now users can carry the USB device anywhere they go. The users is ready to use the key management services upon availability of any machines even if the machine does not belong to them, such as in a desktop in the Internet café, or in a laptop at a friend's place. At this stage, there is no mechanism in place that ties up the owner of the USB to the USB device. If a USB is lost, the user's data in the cloud could become the subject of misuse. Though a simple username/password authentication could be of a potential solution, more sophisticated solutions are provided by the use of biometric based technologies as such we explored in our other paper [13].

B. Data Security

We describe design considerations to protect data in this section. This includes: where to process the key computation to ensure key information leakage does not happen, encryption strategies to protect the data even if the data is intercepted, and data integrity mechanisms to detect from any potential tampering.

1) Computation

In our context, data computation refers to the processing of data from plaintext to scrambled version to avoid adversaries does not learn about the data. As much as it is important to protect the keys that assist computation process, it is equally important to protect the actual computation process itself. The computation needs to be done in a secure environment in such a way that important key information does not leak to any unauthorized users. For example, if the encryption processing takes place at the dishonest host machine, there could be a piece of malicious code hidden monitoring, or worse, stealing the key information used for the computation. The stolen information is then sent to the adversary who remotely controls the malicious code and uses it for the further criminal activities.

We propose the encryption processing to be done only inside our TED (*satisfies the security requirement 1*). In our proposed solution, the encryption processing is performed inside TED by the use of sealing and unsealing operations. Sealing refers to a process of the key management service requesting to the TPM (inside TED) to encrypt user's data. At the time of sealing, the platform information is recorded. Unsealing is a process of the key management service requesting to the TPM to decrypt an encrypted message. Along with the encrypted message, the key management service supplies the platform information. The TPM reveals the plaintext only if the current platform information matches to the platform information provided at the time of sealing. This is to safeguard that the platform remains in the same state in between the encryption and the decryption operations without potentially being modified by malicious code.

However, doing the computation all in the TED comes with performance drawback as the disk space in our TED is small, only 4GB. If the speed is critical, the encryption/decryption can be outsourced to the client given that such computations only take place in the compartment of the Trusted Computing Base (TCB).

2) Encryption

In most cases, data encryption is done by the use of software-based keys which are often stored in a stable memory space maintained by an OS. With the increase of interconnectivity and sophistication of adversaries, these software-based keys become increasingly more vulnerable from remote attacks. Compare to the software bound keys, hardware-based keys are considered to be more difficult to break as the hackers need to steal the actual hardware device to steal the keys contained in the device. Taking this advantage, we use hardware-based keys to assist in data computation process. Even if the encrypted data is hijacked by the man-in-the-middle attack, the malicious attackers will not be able to decrypt the data since the attacker does not know the secret key that is used for the encryption (*satisfies the security requirement R3*)

3) Data Integrity

To preserve high data integrity, our system sends a keyed hash value to be verified. We create a secret key that is only used to secure the communication between two entities. The secret key encrypts the secret message such as cloud user's data. Then the secret key is encrypted by a public part of the key of the receiver to ensure that only the designated receiver can decrypt the secret key. A keyed hash value using HMAC functionality is created by the use of the secret key and the encrypted secret message. The receiving entity verifies the HMAC value to ensure no tampering has occurred during the transmission of the data (*satisfies the security requirement R4*).

C. Platform Assurance

One of the obstacles to take up cloud computing is the lack of transparency. Data owners do not have proper tool supports to ensure their data is not being abused or leaked by the malicious software installed at the cloud server [18]. Similarly, the cloud storage provider does not have a support to know their storage is accessed or utilized by only intended users with legitimate subscriptions.

A promising approach to address this problem is based on Trusted Computing (TC). One of the most innovative ideas realized and supported by TC is remote attestation [18]. The remote attestation allows a remote server (i.e., geographically located away from other server it is interacting) to provide an opportunity to provide an evidence of its platform configurations and operations to other server for verification purpose.

To achieve the goal in providing the transparency, the remote attestation defines mechanisms for two remotely located parties, which are referred as a challenger and an attester, to exchange evidence. The evidence contains two pieces of proof information. One proof is that the messages sent from the challenger are correct, that is, the system the challenger used to produce the messages was honest and produced the message truthfully. The other proof is that the messages are sent by the challenger itself ensuring masquerading has not occurred.

We use the idea of remote attestation to authenticate the identity and platform of the cloud user. The attestation mechanism in our solution prevents any potential abuse of the cloud storage space by malicious adversaries who masquerades to be a legitimate cloud user (*satisfies the security requirement R5 and R6*). It should be noted that the remote attestation runs only one way from the data owner to the cloud storage provider in our solution. Though it would be much more beneficial to the cloud user if a cloud storage provider also provide a piece of evidence (of its server configuration and operations state), it would be unrealistic to enforce such scheme as today's cloud solutions run mostly on a black-box approach.

IV. KEY MANAGEMENT

We describe in details the way our proposed key management service controls a lifecycle of a key, from its inception, data computation, transmission of encrypted data, and deletion of the keys when they are expired.

A. Preliminaries

We use *{M}k t*o denote an encryption of a message *M* using a key *k* under some symmetric encryption algorithm, for example, AES in CBC mode with random initialization vectors. Similarly, *{|M|}k* denotes an encryption of a message *M* using a key *k* under some public-key encryption algorithm, for example, RSA with PKCS encoding standard.

We use the notation *CS* to indicate a cloud storage provider that resides remotely over the public Internet. We use the notation *KEPH* to indicate a key management service that resides inside our TED device. The name Alice is used to represent a cloud user. Alice is also the owner of TED device.

We assume that the public part of the cloud storage provider's keypair, denoted as *CSpub*, has been distributed. Optionally, if a cloud storage server wants to ensure the authenticity of the messages coming from the user, all it needs to do is to verify user's certificate. -

B. Operations

1) Key Creation

In order for Alice to encrypt a message to send to a cloud storage service, what Alice first needs to do is contacting the key management service inside TED with an expiration time and requesting a key. Upon this request, the key management service works with TPM chip to create a unique key pair call *Keph*. An expiration date sent by Alice is attached to the keypair to indicate the period of the key being valid. A key ID is also attached to the key as an identifier. The data tuple *(KeyID, Keph, expiry date)* is created and stored in a database that the key management service maintains. The key is securely held inside the TPM chip. The public part of the key, *Keph*, is returned to Alice along with the key ID.

2) Data Encryption

Alice now encrypts her message (i.e., user data) with a randomly selected secret key *S* to obtain *{M}S*. The secret key *S* is encrypted with *Keph* to obtain *{|S|}Keph*. Alice creates another secret key *T*. Alice encrypts *{|S|}Keph* using *T* obtaining *{{|S|}Keph}T* while *T* is encrypted using CS public key *CSpub* obtaining *{T}CSpub*. Why do we need another secret key *T*? As a common practice, it is more efficient to encrypt the message with a secret key than encrypt it using a public key. We also use the secret key *S* for the data integrity check for *{|S|}Keph*.

Message 1: what Alice sends to the cloud storage provider is:

- *Key ID of Keph*: the ID of the ephemeral key *Keph* that Alice chose to encrypt with.
- *{M}S*: the secret message encrypted with the secret key *S*
- *{{|S|}Keph}T*: the secret key *S* encrypted first by *Keph*, then by another secret key *T*
- *{T}CSpub*: the secret key *T* encrypted by CS public key *CSpub*
- *HMAC(T, {|S|}Keph||KeyID)*: a keyed hash value is created.
- *AIK certificate*: a certificate received from the Privacy CA is sent too.

3) Storing User's Data

When the cloud storage provider receives the encrypted message from Alice, it first authenticates Alice by validating the *AIK certificate*. If Alice's *AIK certificate* is verified, the cloud provider can ensure that the message is truly sent by Alice and the message was computed by a machine containing a legitimate TPM chip.

Once *AIK certificate* is verified, CS decrypts *{T}CSpub* in order to obtain *T*. Using the secret key *T*, the CS obtain *{|S|}Keph*. At this point, the cloud storage provider also verifies *HMAC(T, {S, {|S|}Keph||KeyID)* to ensure no tampering during the transmission. If the HMAC verifies, the encrypted secret message is stored along with Alice's *AIK certificate*.

4) Data Request

When there is a request to download Alice's data from the data centre, the cloud storage provider finds the encrypted message by matching Alice's *AIK certificate*. The CS creates a secret key *J* to encrypt Alice's data *{|S|}Keph* to obtain *{{|S|}Keph}J*. *J* is encrypted by the public part of Alice's *AIK certificate* obtaing *{J}AIKpub*.

Message 2: what the cloud storage provider sends to Alice is:

- *Key ID of Keph*: the ID of the ephemeral key *Keph* that Alice sent along with her data.
- *{M}S*: the secret message encrypted with secret key *S*
- *{{|S|}Keph}J*: the secret key *S* encrypted first by *Keph*, then by another secret key *J*
- *{J}AIKpub*: the secret key *J* encrypted by Alice's AIK public key *AIKpub*.
- *HMAC(J, {|S|}Keph||KeyID)*: a keyed hash value for validation of the message.
- *AIK certificate*: a certificate to be validated by the cloud provider before user's data is being downloaded.

5) Data Decryption

When Alice receives her data, she first verifies the HMAC value to check any potential tampering by the man-in-the-middle attack. Once HMAC is verified, Alice first decrypts *{J}AIKpub* using the private part of her AIK key. By the use of *J*, she obtain *{|S|}Keph* which is then send to the key management service along with KeyID. The key management service uses KeyID to find a matching private *Keph* key. *S* is returned to Alice. Alice uses *S* to decrypt her final message.

6) Key Deletion

We use ephemeral keys (i.e. short-term keys with expiry dates attached to them) to prevent any possibility of a dishonest cloud storage service accessing data after users' subscriptions ended. The key management service does this by periodically checking and deleting the keys with expiry dates elapsed. Any data that has been encrypted by the use of expired ephemeral key become unrecoverable. For example, if the expiry date of *Keph* key is elapsed when Alice sends *{|S|}Keph*, the key management service would not be possible to decrypt *{|S|}Keph*; henceforth, Alice won't be able to decrypt her message since she cannot get the secret key *S*. Similarly, if any copies of Alice's data still remain in the data centre after Alice's subscription ended, the cloud storage service would not be possible to read it. First Alice's data is protected by *Keph* key whose private key never leaves Alice's TED device. Even if the cloud storage provider successfully steals Alice's TED device, the Keph key is already removed by the key management service when Alice's subscription ended.

V. High-Assurance Platform Authentication

When a TED plugs into the host machine, it creates its own trusted environment which isolates its environment from the underlying host machine. Any subsequent process is performed only on the trusted environment created by our TED.

Each TED is equipped with non-forgeable endorsement key which is called as *EK certificate*. The *EK certificate* can uniquely identify each legitimate TED. We assume that the third party certifying authority called Privacy CA knows the list of legitimate holders of *EK certificates*. To preserve its anonymity, the TED never uses *EK certificate* directly to communication with outside world. Instead, it creates a hardware bound *Attestation Identity Key (AIK)* using a key storage structure maintained by each TPM. In addition, our TED measures its platform environment via examination of the hardware BIOS, master boot record, OS by utilizing a set of *Platform Configuration Registers (PCRs)*. The *PCR values* are used to verify if the cloud user, represented by TED, runs on a correct platform state. We assume that the Privacy CA also knows the correct state of each TED. The Privacy CA only certifies *AIK* if the TED presents correct *EK certificate* and *PCR values*.

We explain a high level view of the messages exchanged between TED (represents a data owner who is a cloud user), PCA (represents a trusted certifying authority), and CS (represents a cloud storage provider). The following sequence happens as a part of the platform authentication. The messages produced by the same machine at one time are grouped together using a single numeric number while alphabetic orders added next to the number to denote the sequence of operations.

1a. TED: load protected AIKpvt into TPM

1b. TED: retrieve Quote=sigPCR:nonceAIKpvt

1c. TED: send IdentityReq(cert (EKpub), AIKpub, quote, nonceA)

2a. PCA: validates cert(EKpub) and quote

2b. PCA: send cert(AIKpub)

3. TED: send cert(AIKpub)

4. CS: validate cert(AIKpub)

In step 1a, TED is plugged into a host machine and it collects the *EK certificate*, and generates a public/private AIK pair and a random non-predictable fresh nonce *nonceA*. In step 1b, TED runs a quoting process which measures its platform state using *PCR values*. Then, TED signs the *EK certificate*, *the public part of AIK*, *PCR values*, and *noneA* using the private part of the EK and encrypts these signed data blob using the public part of the Privacy CA key. The encrypted blob is then sent to the Privacy CA as a request to get an *identity credential* as depicted in 1c. In step 2a, the Privacy CA decrypts this blob with its private key and verifies the *EK certificate* and *PCR values*. In step 2b, the Privacy CA then creates an *identity credential* and sends it back to the TED. This credential is a digital certificate containing the *public part of the AIK* together with *nonceA* signed by the Privacy CA private key. The TED uses this credential obtained from CA to authenticate to the cloud provider as depicted in step 3 and 4.

VI. Prototype Implementation

We developed a prototype system to evaluate the feasibility and practical aspects of our proposed solution. The design of our prototype is shown in Figure 4.

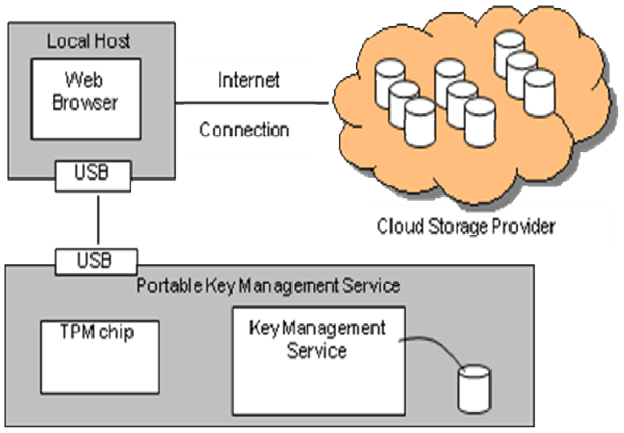

Figure 4. Prototype implementation design

A. System Configuration

The following configurations are used in implemented prototype.

- *Local Host*: We have a desktop machine that represents an untrusted local host. We use Intel Core 2 Duo 6400 with dual processors of 2.13 GHz both with 1.99 GB of RAM running. The Windows XP is run as OS. Our TED uses the Internet connection provided by the local host to communicates with service provided by the cloud storage provider.
- *Platform Authentication Service and Portable Key Management Service*: these are essentially two services added to our existing TED device. Platform Authentication Service runs as the combination of C++/Java applications to communicate both with TPM chip inside TED device and a Privacy CA to obtain an identity credential. A key management

service is a java applications that communicates with TPM chip inside TED to manage keys and assist key computation.

- *Cloud Storage Service:* to avoid any proprietary features of a particular cloud storage service publically available (therefore difficult to port at a later stage), we decided to develop our own cloud storage service to accommodate more generic features. For this, a server machine is used to represent a cloud storage provider. The cloud provider machine also runs the same configuration as the desktop machine above. We developed a simple web application which provides a service that allows the cloud user to upload and download files. We use a MySQL database to simulate a data center and stores user's data there. Implementation of Major Functionalities

1) Trust Extension Device

Our existing TED is extended to provide a number of TPM functionality required by platform authentication and key management as follows.

– TPM CreateWrapKey: creates a new TPM key.
– TPM_LoadKey: loads the newly create key into TPM. Now the key is ready for use.
– TPM_Seal: encrypts the given plaintext with a TPM key. The PCR values needs to be specified.
– TPM_Unseal: decrypted the given cipher with a TPM key. Data is decrypted only when the given PCR values match to the ones supplied at the time of TPM-Seal.
– TPM_Extend: updates a PCR by hashing in a measurement value. We measure a hash value of TED image and supply as the measurement value.
– TPM_Quote: obtains a signed report of the current PCR values.
– TPM_MakeIdentity: create an attestation identity key (AIK)
– TPM_ActivateIdentity: decrypt an AIK certicate obtained from a Privacy CA.

2) Key Management Service

The basic function of our key management service is to create ephemeral keys with tuples <KeyID, Keph, expiration date> and make them available for encryption. The key is then used to encrypt the data in such a way that user's data cannot be decrypted without the aids from the key management service. Another important function the key management service provides is to check the keys with their respective expiry dates. Any keys with the expiration date elapsed are deleted periodically.

– CreateKey: creates an ephemeral key with a given expiry date.
– EncryptData: encrypts the data blob. Utilises the TPM_Seal to store the current platform value along with the keys used for the encryption.
– DecryptData: decrypts the data blob. Utilises the TPM_Unsela to retrieve the secret key.
– DeleteKeys: periodically deletes the keys whose expiration dates have been elapsed.

3) Cloud Storage Provider

As mentioned, we have developed a cloud storage service to provide the most generic features of many cloud storage solutions available in public today. This includes a service to upload and download data. These features are implemented in a simple web application.

– UploadData: cloud user utilizes this feature to upload encrypted data blob. User's data is stored in the database only if the user is authenticated via the use of the platform authentication service.
– DownloadData: cloud user uses this feature to download the data that has been stored in the cloud storage.

B. Observation

We describe preliminary observations on the performance of our proposed system. Compare to the software-based key management, the hardware-based key management seems to be slow. Especially it had an overhead at the initialising phase which took average 2.5 second. The significant overhead happened when TED prepares necessary resources to create and load the first set of root keys and subsequent keys. Also due to the size of our TED, which is housed by 4 GB USB device, the data computation was slow. If TED is equipped with a faster processor and more memory space, this problem is likely to improve.

As reported in [12], our TED does not have its own input and output devices, such as keyboard, mouse, and video cards. This has a vulnerability of having to rely on the local host's input and output devices, which can be untrustworthy. Subsequently, our prototype implementation may potentially suffer from security breach created by keystroke loggers, screen grabbers or other malicious exploit targeted for user interface. We are exploring different ways to mitigate this limitation. The most possible candidates to rectify the limitations include techniques such as encryption-based trusted path [14] or virtual KVM (keyboard, video and mouse) [15].

VII. RELATED WORK

In broad category, there are two schools of thoughts in dealing with data protection for data owner in the cloud.

In the first category, it handles the data protection through the use of third party auditing mechanism. The terms utilizing the concept of public auditability have gained popularity in recent years [6], [22], [23], [24]. In this approach, a third party auditor (TPA) acts as an external service to verify the correctness of the user's data as to whether data confidentiality and data integrity of user's data is preserved. The verification can be done static at a specific time frame or dynamic in real-time. However, this approach has been criticized [6] that they do not support the privacy protection of user's data against external auditors which potential become security breach medium to reveal user's data content accidentally. What we see in these auditing approaches is that they are aimed at detecting breaches whereas our approach is more geared towards the prevention.

Ours and these audit-based approaches can be good complimentary techniques to each other.

More related to our approaches are the ones that use obfuscation techniques to scramble the content of data before transmitting to the cloud so that any unauthorized parties could not learn about the data. The most common techniques used in data obfuscation techniques include such as encryption, fragmentation, and hardware-based Trusted Platform Module [3]. These techniques often used alone or as combined. In [10], the author proposes the concept of using a privacy manager that explicitly handles the obfuscation of the data through encryption. The proposal also utilizes the concept from Trusted Platform Module (TPM) to use it as a hardware-based cryptographic tool. TPM stores the keys and assist the obfuscation process (i.e., encryption) on the user's machine, based on security policy, before sending the encrypted data to the cloud storage provider. In [27] and [28], authors propose to use fragmentation technique to slice up a large chunk of data into smaller pieces. Each fragment is encrypted using a unique key. Each encrypted fragment is then sends to the cloud. The details of the management of the keys and the protection of the keys are not addressed in the paper. Hardware tamper-proof token based approach was briefly explored in [16]. In this approach, TPM is used to store the keys as well as to store the keys in the dedicated sealed environment. The key computation also takes place in the sealed environment. This design approach makes it more self-resilient from potential attack since the keys and the computation are both better protected in the sealed area. Though it has merits, the proposed solution is implemented in the public cloud and would suffer a number of problems. Most significantly, the availability will suffer if one of the clouds, containing either the key or the data, goes down and the user would be left not being able to access their data when they need. In addition, the public cloud is insecure, especially to store secrets. There's number of growing concerns for the cyber attacks that are specially geared towards public cloud. Furthermore, the key management aspects were never mentioned.

VIII. Conclusion

We have proposed a portable key management service that centralizes all the expense and expertise of maintaining a large number of keys, and yet the keys are highly secure and available. Our key management service employs to encrypt user's data before they are sent to the cloud while the secret keys used to aid the encryption is securely held inside a tamper proof hardware device. This effectively provides a mechanism to utilize the cloud storage in a secure manner while protecting the keys from the cloud storage services.

We address a number of important design considerations to protect the secret keys, user's data, and the platform.

All keys managed by our key management service are secure as the keys are stored in a designated sealed storage area within a tamper-proof hardware device. This ensures that the general stealth of unique keys especially by the use of remote Internet connections is much more difficult. In addition, the keys created by our key management service are ephemeral in nature. The ephemeral keys have temporary lifetime and deleted when the expiration date attached to each key is elapsed. Any data encrypted by the ephemeral key whose expiration date is passed become unrecoverable.

Our key management service is highly available as they are not locked into any particular machine. Rather, our key management is implemented in a portable USB device that users can carry with them all the time and use it at any time as needed. In addition, our solution offers a high assurance platform authentication where the cloud user can be authenticated by the cloud storage server to prevent its storage space be wasted by malicious attackers. By verifying the evidence sent by the user, the cloud storage provider ensures that the requests actually came from the legitimate user's platform free from running any malicious software.

In the near future, we plan to formalize the proof of our security protocols using a security verification tool such as SPIN or fs2pv [26]. As the mobile devices such as smartphone and tablet PCs becoming a big player in the cloud computing, we have a plan to port our proposal into these devices. Also, we plan to incorporate our solution to work with a commercially available cloud storage service such as Amazon S3.

References

[1] Amazon Simple Storage Service (S3): aws.amazon.com/s3/

[2] Trusted Computing Group (TCG): www.trustedcomputinggroup.org

[3] TPM: www.trustedcomputinggroup.org/groups/tpm/

[4] Australian Government Report, Cloud Computing: Trends and Threats: http://www.aic.gov.au/documents/C/4/D/%7BC4D887F9-7D3B-4CFE-9D88-567C01AB8CA0%7Dtandi400.pdf, 2010

[5] Cloud Security Alliance, Top Cloud Computing Threats: https://cloudsecurityalliance.org/topthreats/csathreats.v1.0.pdf

[6] C. Wang, Q. Wang, K. Ren, and W. Lou: Ensuring data storage security in Cloud Computing. IWQoS 2009: 1-9

[7] C. Wang, Q. Wang, K. Ren, and W. Lou: Privacy-Preserving Auditing for Data Storage Security in Cloud Computing, InfoCom 2010, 525 -533

[8] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia: Above the clouds: A Berkeley view of cloud computing. University of California, Berkeley, Tech. Rep. USB-EECS-2009-28, Feb 2009.

[9] R. Neisse, Y. Holling, A. Pretschner: Implementing Trust in Cloud Infrastructure. CCGrid 2011: 524-533

[10] S. Pearson, Y. Shen, M. Mowbray: A privacy manager for cloud computing. CloudCom 2009: 90–106

[11] L. Chen, R. Landfermann, H. L¨ohr, M. Rohe, A.-R. Sadeghi, and C. St¨uble. A protocol for property-based attestation. STC'06

[12] S. Nepal, J. Zic, D. Liu, J. Jang: Trusted Computing Platform in Your Pocket. TrustCom 2010: 812-817

[13] J. Jang, H. Hwang, S. Nepal: Biometric Enabled Portable Trusted Platform Module: TrustCom 2011:436-432

[14] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh. Terra: A virtual machine based platform for trusted computing. SOSP:2006; 193–206.

[15] R. Meushaw and D. Simard. NetTop: Commercial technology in high assurance applications (2000). http://www.vmware.com/pdf/TechTrendNotes.pdf

[16] A.-R. Sadeghi, T. Schneider, and M. Winandy: Token-based cloud computing. in TRUST, ser. Lecture Notes in Computer Science, A. Acquisti, S. W. Smith, and A.-R. Sadeghi, Eds., vol. 6101. Springer, 2010, pp. 417–429.

[17] R. Perlman: The Ephemerizer: Making Data Disappear. Sun Microsystems Technical Report SMLI TR-2005-140 February 2005

[18] R. Chow, P. Golle, M. Jakobsson, E. Shi, J. Staddon, R. Masuoka, J. Molina: Controlling data in the cloud: outsourcing computation without outsourcing control. CCSW 2009:85–90

[19] T. Mather, S. Kumaraswamy, S. Latif: Cloud Security and Privacy. O'Reilly, 2009.

[20] M. Jensen, J.O. Schwenk, N. Gruschka, and L.L. Iacono:On Technical Security Issues in Cloud Computing: CLOUD-II 2009:109-116.

[21] M.A. Vouk: Cloud computing issues, research and implementations. ITI 2008:31-40.

[22] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song: Provable data possession at untrusted stores: CCS2007: 598–609.

[23] A. Juels and B. S. Kaliski, Jr.:Pors: proofs of retrievability for large files: CCS2007:584–597.

[24] E.-C. Chang and J. Xu: Remote integrity check with dishonest storage server. ESORICS'08:223–237.

[25] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn. Design and implementation of TCG-based integrity measurement architecture. In Proceedings of the USENIX Security Symposium, 2004.

[26] Fs2Pv, Security Protocol Verifier: http://research.microsoft.com/en-us/downloads/d54de3ef-085e-47f0-b7dc-8d56c858aba2/default.aspx

[27] J. Yao, S. Chen, S. Nepal, D. Levy, and J. Zic. "TrustStore: Making Amazon S3 Trustworthy with Services Composition". *CCGRID* 2010, pp. 600-605.

[28] S. Nepal, C. Friedrich, L. Henry, S. Chen: A Secure Storage Sevice in the Hybrid Cloud, UCC 2011, pp 334-335.

[29] S. Kamara and K. Lauter, "Cryptographic cloud storage," in RLCPS, January 2010, LNCS. Springer, Heidelberg.

[30] S. Nepal, J. Zic, H. Hwang, D. Moreland: Providing Mobility and Portability of Trust in Cooperative Information Systems, CoopIS, 2007, pp 253-271

[31] S. Nepal, J. Zic, J. Jang and D. Liu: A Mobile and Portable Trusted Computing Platform, EURASIP Journal on Wireless: 7(1):75-94, 2011.