2011 International Joint Conference of IEEE TrustCom-11/IEEE ICESS-11/FCST-11

CLOUD SHREDDER:

Removing the Laptop On-Road Data Disclosure Threat in the Cloud Computing Era

Nan Zhang
Graduate University of Chinese Academy of Sciences, Beijing, China
Email: zhangnan@is.ac.cn

Jiwu Jing
Graduate University of Chinese Academy of Sciences, Beijing, China
Email: jing@is.ac.cn

Peng Liu
College of Information Science & Technology, Pennsylvania State University, University Park, PA, USA
Email: pliu@ist.psu.edu

***Abstract*—Data Disclosure due to laptop loss, especially in travel, is a top threat to businesses, governments, and non-profit organizations. An effective protection against this threat should guarantee the data confidentiality, even if the adversary has physically possessed the laptop. Current technology does not satisfy this requirement. This paper proposes a novel approach to remove the threat under the emerging condition of ubiquitous internet access and cloud computing. We name this approach "Cloud Shredder", implying that the confidential files are shredded and hidden in the semi-trusted cloud storage service. Cloud Shredder is a generic and transparent security service that allows legitimate user access the files in exactly the same way as with commodity file systems, whereas the attackers only get meaningless junk even if they have obtained every byte on the hard drive. Rather than the traditional encryption-based protection, Cloud Shredder limits the attacker's opportunity in a short time window. We implemented a prototype that is compatible with the typical cloud storage service, Amazon S3, and supports two popular document applications, Acrobat Reader and Open Office. Our experiments show that the influence on file access performance is reasonable and should not ruin the user experience. Cloud Shredder is also applicable to smart phone, netbook and other computing devices with internet connection.**

***Index Terms*—data protection; cloud computing; laptop theft; secret sharing**

I. INTRODUCTION

For successive years, the loss and theft of laptops and the subsequent data disclosure have been ranked in the top three most frequently occurring incidents that cause serious troubles for companies and individuals in a wide range of industries [1], [2]. That travelers' habits of using their laptops in airports, railway stations and other public places increase the possibility of laptop loss. A recent study [3] claims that 12,000 laptops are lost, missing, or stolen each week at U.S. airports. If the lost laptops contain confidential data, there would be a serious risk not only of economic interest, but also of social reputation.

Most laptop theft cases are non-premeditated. They are usually resulted from the laptop owners' carelessness. The thieves in these cases mainly aim to sell the stealing on the black market for money. They are not interested in whether the sensitive data exists. However, the hard drives may be disassembled from the laptop and sold to some "professional" attackers to exploit the value of the data on them. These attackers may have stronger skills and computing resources than common users to bypass or compromise the security protection of the hard drive.

The representative scenario of carrying sensitive data in travel is that a businessman visits a client in another city to give presentation with slides containing confidential information. In the client's office, he or she needs to present the slides. During the trip, he or she also needs to use the laptop to deal with other affairs, and occasionally read through or even revise the slides. In the whole process, the confidential slides are expected to be protected from illegitimate users' access. When the laptop is stolen, the adversary physically possesses the laptop, the data confidentiality should still be guaranteed.

The first solution coming to mind is encryption. The concept "Full Disk Encryption" (FDE) has been proposed and there are already a few commercial product based on this idea, e.g., BitLocker [4] [5] for Microsoft Windows. However, FDE is not competent in the real-life cases of laptop theft. The attackers can take out the hard drives and mount them as the slave disks on their own computers to bypass all the authentication mechanisms on the original computers. Then they can freely access the encrypted data on the hard drive and carry out the off-line attack, such as brute force attack and dictionary attack, without time pressure or restriction on trial times. In FDE, the length of the encryption key is chosen on the basis of "computational security", assuming the attackers only have currently common computing resources. However, this assumption may not apply to the adversaries who are intended to recover the data on hard drive. Even if the user uses a longer key, he still cannot change the "fate" of lost files. As long as the attacker has sufficient resources and time, he can always do brute force attack on the FDE storage key and successfully decrypt the cipher text of interest. In other words, whether the lost information can be kept secure is no longer under the control of the user; it is under the control of the attacker's desire and resources. Once the laptop is lost, the user can do nothing but feel anxious and helpless.

Our goal is to get the control back. Our approach is proposed to guarantee that lost files are useless (for the attackers to get useful data) except for a rather small time window. Within this time window, our approach is still vulnerable to information leak. However, when the time window expires, we guarantee it is no longer possible to decrypt/retrieve any useful information from the lost files. Our approach has this unique advantage because of the following idea. The data stored on the hard drive should not be the cipher text of any clear text.

978-0-7695-4600-1/11 $26.00 © 2011 IEEE
DOI 10.1109/TrustCom.2011.221

For example, we can always split one file X into the XOR of two shares, Local Share (LS) and Remote Share (RS). If we only store LS on the hard drive, and keep RS from staying on hard drive in any cases, then it will be impossible to restore X no matter how many resources the attacker has. Our approach focuses on how to protect RS and deliver RS to the user whenever the user wants to access file X.

In particular, we proposed to use the emerging cloud service to achieve these two goals: (1) RS delivery and (2) RS protection. We found cloud service can provide high quality RS delivery service through a universal identifier (e.g., a pathname in Amazon S3). We instrumented document processing softwares in common use, e.g., Acrobat reader and Open Office, to transparently merge LS and RS. In this way, the usability is not affected. Regarding RS protection, the cloud service also has a good property by authenticating each request of downloading. We assume this authentication itself can not be compromised by the attacker in the scope of this paper. The attacker's only choice is to try on cloud secret credential (e.g., SAK in Amazon S3). To defeat this attack, our approach aims to erase RS in the cloud before the attacker breaks in. That is, as long as the user realizes the laptop disappearance and react by instructing the cloud service to erase RS soon enough, our approach will achieve perfect security. Here we define two concepts. **Vulnerability Time Window** (also mentioned as "time window" for short) is the time span from when the laptop loss/theft takes place to when the user reacts. **Essential Attacking Time** (EAT), is the least time needed by the attackers to find a way to obtain the RS in the cloud. Obviously, as long as the EAT is longer than the Vulnerability Time Window, the file X will be safe. In our implementation, we adopted techniques to guarantee that the EAT is longer than the Vulnerability Time Window in practise. This means our approach has excellent practical security.

Our work has three contributions: (1) A novel data protection approach under the new condition of ubiquitous internet access and cloud computing, which have already partially achieved and will realize in the foreseeable future. (2) An implementation which is compatible with a typical cloud storage service, Amazon S3, and supports two popular document applications, Acrobat Reader and Open Office. (3) The performance evaluation based on a series of experiments in the field.

The rest of this paper is organized as follows: Section II summarizes related work. Section III presents a panoramic view of the Cloud Shredder system. Section IV describes in detail the implementation of each component in the prototype system. Section V explains the principles of the splitting method and presents two examples. Section VI analyzes the security against a real-life attack. Section VII thoroughly evaluates the system's performance in the field. Section VIII discusses the prototype and looks ahead at future work. Section IX draws the conclusion.

II. RELATED WORK

A. Full Disk Encryption

One popular technology used to protect the sensitive data in the hard drive is Full Disk Encryption (FDE) [6] [7], which encrypts every sector on the hard drive and supports Pre-Boot Authentication (PBA). FDE intercepts all hard drive "read" and "write" requests from the operating system, and uses certain symmetric algorithms with a Storage Key (SK) to encrypt every sector of data when the OS writes a file to the hard drive, and decrypt every sector of data into memory when the OS reads a file from the hard drive. However, since the attacker can carry out off-line attack on the cypher text on hard drive, the data security of FDE just depends on the secrecy of the disk encryption key, which is protected by either a password (or PIN) or an external device, such as a USB key. In the first case, the user-memorized password cannot be long and complicated enough to resist dictionary attack or brute force attack. In the second case, the USB key introduces new risk. Its small size makes the USB key easily be misplaced or left behind and then makes the legitimate users not use their laptops as well. Therefore, in practice, many people choose to carry the USB key along with the laptop in the same bag or case. The result is that the laptop thieves have a large chance to obtain the USB key at the same time. Here comes a dilemma: the short secret (e.g. user-memorized password), under the off-line attack, could not provide enough confidentiality, whereas the long secret (e.g., USB key) does not really improve the confidentiality sometime but possibly harm it.

B. Cloud Storage

One option for preventing laptop thieves from getting sensitive files is to never store them in the laptop, but to store them in another location. With widespread Wi-Fi hot spots and third-generation mobile communication technology available, travelers can access the online services in the "cloud" almost any time and anywhere. Among diverse Cloud Computing Services, the Cloud Storage Service provides individuals and companies an economic solution to host files online and access them from different places with satisfying, reliable and consistent performance. The most well-known infrastructure of the Cloud Storage Services is Amazon Simple Storage Service (S3)[8]. There are already a few cloud storage products based on S3 and other infrastructures in the market. Some of them, such as Drop Box[9], emphasize on backup, and others, such as Rapid Share[10] focus on web sharing. However, the major concern about Cloud Storage is its security. Clients are not yet likely to entrust their sensitive data to another company because they fear that the Cloud Storage Services may be curious about the content of the files and use them illegally. Even encrypting the files may not remove the risk completely, because the company who operates the Cloud Storage Service have abundant computing resources and hence have the possibility to crack the cipher. Therefore, the files are not considered as safe if outsoured to the cloud.

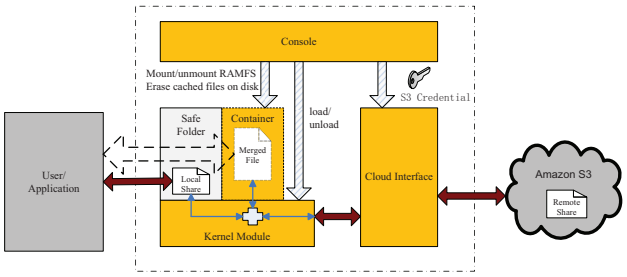

Fig. 1. Architecture of the Cloud Shredder

C. Capture-Resilient Device

MacKenzie et al. presented a technique[11] by which a device that performs private key operations (signatures or decryptions) in networked applications, and whose local private key is activated with a password or PIN, can be immunized to off-line dictionary attacks in case the device is captured. This technique forces the attacker to verify password guesses at an online server, thereby turning off-line attack to the captured encryption device into an online attack to the server which can be detected and stopped by the server. This technique protects the networked devices with short secrets.

III. Overview

A. Assumptions

To simplify the complicated situations in practice, we make four assumptions: (1) Theft cases only happen when the laptops are temporarily out of the owner's control, namely not in use. At that time, the laptop should be shut down or at least the user should have logged off. (2) Whenever needed, the access to cloud storage service is always available by wired network, Wi-Fi, or 3G mobile network. We will not consider the exceptional cases such as connection absence, discontinuity or transmission failure in this paper. We believe this assumption is reasonable because the traveler just occasionally use the confidential files on-road and will prefer to find a place with available internet, such as airport lounge, cafe shop or hotel. (3) The Cloud storage service is considered to be "semi-trusted", i.e. although it may be curious about the content of the files, it will not collude with the other adversaries. (4) The most frequently carried/used confidential files in trip are documents and slides, whose sizes are seldom very large. We do not consider the exceptionally large ones in this paper and limit the files smaller than 5M byte in our experiments, but will discuss this issue in the future work section.

B. Architecture

The Cloud Shredder is essentially a transparent, in-kernel security service. It ensures the confidentiality of the protected files when the laptop is in the adversary's hands. As shown in Figure 1, the Cloud Shredder has five components: Safe Folder, Merged File Container, Kernel Module, Cloud Interface, and Console. The Safe Folder looks like a common file directory except that whenever the Cloud Shredder is not activated, the folder only contains the local share of each protected file. From the local share, none of the file's content can be discovered by the adversary who steals the laptop. Likewise, without the local share, the remote share does not give the possible adversary in the cloud service any of the file's contents either. When the Cloud Shredder is activated by valid password, the protected files could be accessed successfully. In case the legitimate user is accessing a file X, a specific remote share of X will be downloaded by Cloud Shredder from the remote storage cloud through Cloud Interface. Once the remote share is in the RAM, Cloud Shredder will properly merge the two shares together and restore the whole information contained in X and temporarily store X inside the Merged File Container. Then the merged file (also called the "real" file in this paper) will be transparently used. In Figure 1, the dash line arrow depicts the user's illusion of handling the merged file directly.

If the user reads file X without making any modifications, the merged file will never stay on the disk (i.e., traceless on disk). When the user is finished with X and logs off, the Cloud Shredder will just release the memory used by the merged file. If the user modifies file X, whenever the user saves the intermediate changes to X, Cloud Shredder offers two options: (A) Split the current version of X into the new local share and the new remote share and send the remote one to the cloud, while storing only the local share on the hard drive. (B) Do not send anything to the cloud, but store the modified merged file temporarily on the disk. When the user closes X, the latest version of the merged file will be split and will replace the previous local and remote shares. If option A is chosen, the merged file is as traceless on the disk as the read-only case, but every time the file is saved, the user has to wait for the splitting and uploading.

Besides the Safe Folder, Cloud Shredder is comprised of four components: The Kernel Module provides the wrappers of the file-operation-related system calls to implement the splitting and merging. The Cloud Interface handles the communication with the storage cloud service. The Merged File Container is the logical container for transient merged files. The Console controls the above three components.

IV. Implementation

We implemented a prototype of Cloud Shredder on Linux and utilized Amazon S3 as the semi-trusted Cloud Storage Service.

A. Kernel Module

In order to be friendly to users, especially non-technical users, Cloud Shredder should not change the users' experience of the file operations. Therefore, we wrap the file operations. In the wrappers, for the files outside the Safe Folder, the system's original operations are executed, whereas for the files inside, extra operations are carried out before and after the original ones. In other words, Cloud Shredder splits and merges files transparently.

In Linux operating system, six layers are involved in file operations, as shown in Figure 2. APIs provide the interfaces

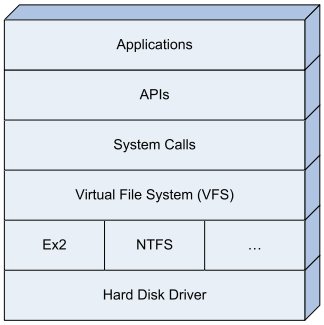

Fig. 2. Layer model of Linux's file operations

to applications and call the corresponding system calls. System calls are implemented in kernel and carry out the file operations through an abstract layer named Virtual File System (VFS). The VFS maps the file operation routines of different concrete file systems, such as ext2 and NTFS, to a series of common interfaces. With VFS, all concrete file systems are handled the same way in the view of system calls and upper layers. Wrappers can be inserted in three places. The first one is the API layer. Because APIs are implemented in library files in operating systems, the original library files needs to be substituted. This leads to difficulties when Cloud Shredder is deployed on the user's laptops. Another possible place to insert wrappers is the concrete file system driver. This leads to the dependency on the specific file system type of the laptop and also influence the deployment. The third place, the system call layer, is located between the above two. We insert the wrappers of system calls by hooking the system call table, i.e., replacing the addresses of certain original system calls with the address of its wrapper. In this way, it is easy to hook and restore system calls just by loading and unloading a kernel module. Therefore, neither the recompiling of the kernel nor reinstallation of the OS is necessary–Cloud Shredder could be deployed and controlled very effectively. In our prototype, we hook system calls for five basic file operations: open, close, write, rename, and unlink (delete). The first three are the basic operations needed by any file editing. The last two are needed when dealing with temp files during editing. The compatibility with Acrobat Reader and Open Office series softwares are well tuned during the implementation of the kernel module.

B. Cloud Interface

We store the remote shares in the "cloud". In the prototype, we adopt the Simple Storage Service (S3)[8] powered by Amazon Inc. Amazon S3 stores objects in buckets. Each object's key, a character string, is the unique identifier of the object within each bucket. Every user is assigned an "security credential", including an Access Key Id (AKI) and an Secret Access Key (SAK). The AKI is similar to username or id and is used publicly. The SAK is used to calculate a HMAC-SHA1 signature to be attached in every request sent from the user to S3. Only the request with a valid signature is accepted and processed by S3. The user's SAK is a secret and should not be disclosed to a third party. In our prototype, every user has been assigned an S3 account and a bucket. All the remote shares of this user's real files are uploaded to this bucket and stored as objects. For each laptop equipped with Cloud Shredder, the corresponding bucket address is configured in the system and the SAK is encrypted with a secret only the laptop owner knows, which is considered as the Cloud Shredder password. Each time the laptop owner activates Cloud Shredder, the password is required. Only with the correct password, the SAK could be deciphered and used for downloading the remote shares from S3. In this way, no shadow file of the password exists on the hard drive. Different from shadow file, the SAK cypher text could not be used by the attacker, because the SAK is just an alphanumeric string without any structure. The only way to verify an alphanumeric string is the right SAK or not is sending a query to the cloud server. Therefore, the off-line attack to the SAK is not practical.

In order to increase the "Essential Attacking Time", we do not directly use the password to encrypt the SAK, but use the password as the input of a key strengthening function and derive a longer key.[12][13] The key is never retained on the hard drive, so the attacker can not obtain it without deriving from the password again. The extra time cost (e.g., 1 second) by once key derivation operation is acceptable when the legitimate user activates Cloud Shredder, but the extra time cost will be accumulated to a significant value when the attacker carries out brutal force trial on the password.

C. Merged File Container

Logically, all the merged files are temporarily maintained in the Merged File Container. But the physical location of these files depends on a user's choice when closing a modified files. When every file in the Safe Folder is opened, the merged file is stored within physical memory. Once a save command is given, the merged file may be kept in physical memory (option A) or be stored to the hard disk (option B).

Within the physical memory: For the option A, the Cloud Shredder should ensure that none of the merged file content appears on the hard disk. Hence, we use a special file system, RAMFS [14], to contain the merged files. RAMFS is a dynamically resizable RAM-based file system. All files stored in RAMFS only exist in physical memory. When a file system is mounted as RAMFS, it seems like a blank directory where any file or subdirectory could be created and accessed like the ones on the hard drive. After it is unmounted, all the files and subdirectories disappear, making the merged files traceless because they are never written to the hard drive.

On the hard drive: For the option B, the Cloud Shredder should effectively delete the merged file after log-off. Since normal deleting does not wipe off the data on the hard drive, we need to intentionally overwrite the corresponding hard disk sectors with patterns of meaningless data. Security is ensured by doing multiple "layers" of random overwriting. This is a mature technique and widely used in a lot of commercial software. Most of them conform to or exceed the standards of the Department of Defense and the National Security Agency

[15]. In our prototype, we make use of a Linux command, `shred`.

D. Console

The last component, Console, is the controller of the Cloud Shredder system. It has three different jobs: (1) When the legitimate user activates Cloud Shredder with password, Console deciphers the S3 SAK, loads the kernel module into Linux kernel and mounts the RAMFS to enable the Merged File Container. Then all the sensitive files "appear" to the user. (2) After the Cloud Shredder has been activated, the Console monitors the system and forces the current user to log off when either the laptop's lid is closed or the system is inactive for a certain period (15 minutes for our prototype, configurable by user). (3) When the user logs off, the Console deactivates Cloud Shredder by unloading the kernel module, un-mounting the RAMFS and wiping off the merged file cached on the hard drive. At this time, all the sensitive files "disappear".

V. Splitting and Merging

In Cloud Shredder, splitting and merging are a pair of inverse operations. Splitting operations separate the sensitive files into two parts on the principle that an adversary cannot recover the real file with only one part. Meanwhile, the legitimate user, who has access to both parts, can recover the real file easily by merging them. In other words, the real file can be reconstructed when and only when both parts are available. Distributing a secret among a group of n participants in a way that at least t participants together can reconstruct the secret is called the (t, n) secret sharing [16]. Therefore, the splitting in the Cloud Shredder is actually a special case of the secret sharing scheme, in which t=n=2.

In information theory, entropy (noted as H) represents the difficulty of finding the information in a random vector (we consider a file to be a random vector). The greater the entropy is, the greater the uncertainty of the vector is. Conditional entropy $H(A|B)$ represents the difficulty of finding out A when B is already known. The larger $H(A|B)$ is, the less is A inferred from B. The Cloud Shredder splits a real file X into a local share L and a remote share R, so that the information in S is shared between L and R. When the laptop is stolen, the adversary obtains L. $H(X|L)$ is the difficulty for the adversary to find out X. The larger $H(X|L)$ is, the harder it is to infer X from L. Similarly, the larger $H(X|R)$ is, the more difficult it is for the adversary in the semi-trusted storage cloud to infer S from R.

The theory of secret sharing tells us that $0 < H(X|L) \leq H(X)$ and $0 < H(X|R) \leq H(X)$. In the "Perfect" Secret Sharing (PSS), X cannot be reconstructed from only one share at all. In other words, no information in X can be learned from a share, so $H(X|L) = H(X|R) = H(X)$. Otherwise, in the "Non-perfect" Secret Sharing (NSS), a file's shares contain partial information about it, so $0 < H(X|L) < H(X)$ and $0 < H(X|R) < H(X)$. In this section, we discuss the splitting method for both two categories.

A. XOR-method

The simplest PSS scheme is by a bitwise XOR operation[17]. We call this the XOR-method. In splitting, we consider X as a byte stream, generate a random byte stream of the same length, and execute bitwise XOR operation on the two streams byte by byte. The result, a new byte stream of the same length, is uploaded to the storage cloud as the remote share R. The random byte stream is stored in the Safe Folder as the local share L. In merging, we conduct bitwise XOR operation on the local share and the downloaded remote share byte by byte, and obtain X. Because L is a random byte stream, $H(X|L) = H(X, L) - H(L) = H(X) + H(L) - H(L) = H(X)$. This approach makes the local share meaningless junk without the remote share, and vice versa. The XOR-method is the securest method because it is equivalent to encrypting X with a one-time pad of the same length [16]. However, its drawback of deterioration in performance cannot be ignored. In PSS, the remote share is at least as long as the real file. When the real file is large, too much time is consumed by the transmission. Hence, we propose the second method in the category of NSS.

B. Ratio-method

The NSS scheme proposes a trade-off between the level of security and share size. Although it is not as secure as the PSS's information-theoretic security, NSS can still provide enough security on the computational basis. There is already a lot of research on the NSS scheme, especially the method to make "shorter shares," such as the Information Dispersal Algorithm [18]. Here we propose our own algorithm, called the Ratio-method: Firstly, transform the real file X into X' using a function F, Secondly, split X' into two parts of not equal length, and and use the smaller part as the remote share R and the longer part as the local share L. In this way the transmission of R via internet will consume less time than using the XOR-method. The reverse transformation F^{-1} should make this effect: any bit of X' influences every bit of X, in other words, to recovery X, the whole X' should be known. This is just the "diffusion" property of a good block cipher [19]. Therefore we could use a symmetric encryption algorithm to play the role of F. Here the encryption key is just a non-secret parameter of the function. We define the proportion of the remote share size to the real file size as the Remote Ratio (RR). The smaller RR means that the Ratio-method is more efficient but of less security. If we set RR=1/N, we pick one from every N blocks (block cypher divides the clear-text file into blocks) and assemble these picked blocks as the remote share. For a file X of L-byte, the remote share will be L/N-byte long. If X is as large as several files, which is common in the scenario we discuss in this paper, and N is chosen as 32, which is shown acceptable in performance in our experiment, the length of remote share will be significantly longer than the length of encryption key in symmetric cipher (usually 128-bit, [20]). Therefore, using Ratio-method, the Cloud Shredder is still remarkable securer than FDE.

TABLE I
RISKS IN THE WORKING CYCLE OF CLOUD SHREDDER

	RAM		Hard drive		
	S3 SAK	Merged Files (Container A)	Merged Files (Container B)	S3 SAK	Local shares
activated	○	–	–	●	○
open file	○	○	–	●	○
save file	○	○	○	●	○
close file	○	○	○	●	○
deactivated	–	–	–	●	○
log off	–	–	–	●	○

VI. SECURITY ANALYSIS

Since the focus of this paper is to propose the novel approach but not the design of splitting algorithm, we analyze the security of the whole system by assuming the splitting algorithm is secure. (The research on the algorithm design and evaluation will be the future work.) As stated in the introduction section, this research just address the non-premeditated theft cases, in which the attacker's behavior to the sensitive data in the stolen laptop has the following four characters: (a) The laptop loss/theft only occurs when the laptop is not in use. Since Cloud Shredder will be deactivated when user logs off and we provide automatical log-off ability in implementation, we just discuss the cases occur when the Cloud Shredder is deactivated. (b) The attacker can fully control and access the hard drive of the stolen laptop. (c) Although the public Wi-Fi service is considered not secure and may be vulnerable to eavesdropping, the thief/attacker who is not premeditated to exploit sensitive data in certain laptop do not eavesdrop the communication between that laptop and cloud service in advance to collect the copies of remote shares. (d) The Cloud server's authentication mechanism is secure, and will not collude with outside attackers, therefore the attacker cannot access any files in the cloud storage without providing a valid SAK matching the laptop owner's AKI.

Table I depicts the places where the sensitive data exist in different stages of the working cycle of Cloud Shredder. The symbol "○" and "●" represent respectively that the data exists in the given stage in the form of clear text and cipher. The symbol "–" represents that the data does not exist in the given stage. After Cloud Shredder is activated by user, the S3 SAK resides in RAM all the time until it is deactivated. Once a file in the Safe Folder is opened, the corresponding merged file resides in the Container A within RAM. After an intermediate version is saved, the merged file remains in the Container B on the hard drive. When the Cloud Shredder is deactivated, either by user manually or by the Console module automatically, the merged files in Container A are released and the merged files in Container B are wiped off securely.

The attacker can not obtain sensitive data from hard drive or RAM. The attacker only obtain the laptop when it is not in use. At that time, Cloud Shredder has "closed" the Safe Folder and erased all the cached content. The attacker can only obtain from the laptop the local shares and encrypted S3 SAK but neither merged files nor clear-text S3 SAK. Since inferring the real files only from the local shares is impossible, Cloud Shredder's security depends on whether the attacker is able to obtain the remote shares.

The attacker can not obtain the remote shares in transmission. The remote shares do transmit in a not well-protected channel between the laptop and cloud service when Cloud Shredder works. However, the thieves/attackers do not have chance to eavesdrop. Before they obtain the laptop, they do not know the owner's username and SAK and even whether there is a laptop protected by Cloud Shredder around them. Eavesdropping all the wireless communication and hope to collect the right remote shares by chance is not beneficial for the thieves/attackers, so they will not do that. (If we extend the above attacking model and assume the eavesdropping, a SSL channel between the laptop and cloud server will well protect the remote share in transmitting.) Once the laptop is lost, the transmission of remote shares will not take place any more. Therefore, the remote shares are not at risk of disclosure.

The attacker hardly obtain the remote shares from the cloud storage service. To access the remote shares, the attacker must request to the S3 cloud storage with the valid SAK. The attacker has two choices: (1) directly try the SAK by exhaustion and (2) guess the password and derive possible key following the derivation algorithm and use it to decrypt out the SAK. No matter which way the attacker choose, he/she needs to succeed within the vulnerability time window, which is usually as short as several minutes. Otherwise, the remote shares will be erased forever under the instruction by the user. Unfortunately, both the two ways are time consumed. For the first way, the S3 SAK, a 40-alphanumeric-long string, is too long for exhaustion. For the second way, the key strengthen algorithm significantly increases the time cost by generating each key to try. In our implementation, we use 8-digit password and a key deriving algorithm which costs 1 second for each key. Then, exhaustion of all possible key will cost 10^8 seconds, namely many days. In either way of the above two, the attacker hardly succeed the compromise of SAK within the vulnerability time window. For the extreme condition, that the attacker has the ability to do the exhaustion on the SAK quick enough, the attacker still can not effectively exploit the vulnerability time window, because the cloud service will not response quickly enough as attacker needs. Not to mention that the cloud server may lock the account after a number of failed trails. As a result, the "Essential Attacking Time" should not be shorter than the vulnerability time window.

Compared to FDE, Cloud Shredder is securer for two reasons: (1) It forces the attacker to attack the cloud storage, an online service, which is much harder to compromise than the encrypted hard drive which the attacker has full control. (2) It restricts the attacker's success opportunity by a limited time window, whereas FDE give the attacker unlimited time. Because of the above two advantages, Cloud Shredder is secure enough to protect the data in on-road laptop loss and theft cases in practice.

TABLE II
PERFORMANCE IN AN OFFICE (UNIT: SECOND)

(a) XOR-method SEL

	500k		1M		3M		5M	
	Mean	σ	Mean	σ	Mean	σ	Mean	σ
T_d	0.39	0.09	0.73	0.08	2.09	0.11	3.45	0.16
T_m	0.09	0.02	0.17	0.05	0.54	0.03	0.91	0.03
SEL	0.48	0.10	0.90	0.11	2.63	0.13	4.36	0.19
T_d/SEL	81.0%	–	81.2%	–	79.5%	–	79.1%	–
T_m/SEL	19.0%	–	18.8%	–	20.5%	–	20.9%	–

(b) Ratio-method SEL

	500k		1M		3M		5M	
	Mean	σ	Mean	σ	Mean	σ	Mean	σ
T_d	0.21	0.01	0.22	0.02	0.26	0.03	0.33	0.04
T_m	0.05	0.01	0.10	0.02	0.31	0.02	0.50	0.02
SEL	0.26	0.02	0.32	0.04	0.57	0.05	0.83	0.05
T_d/SEL	80.8%	–	68.8%	–	45.6%	–	39.8%	–
T_m/SEL	19.2%	–	31.2%	–	54.4%	–	60.2%	–

(c) HEL

	500k	1M	3M	5M
HEL_0	1.4	1.6	2.0	2.4
HEL_1	1.9	2.5	4.6	6.9
HEL_2	1.7	1.9	2.6	3.2
improv.	10.5%	24%	43.5%	53.6%

TABLE III
PERFORMANCE AT AN AIRPORT (UNIT: SECOND)

(a) XOR-method SEL

	500k		1M		3M		5M	
	Mean	σ	Mean	σ	Mean	σ	Mean	σ
T_d	3.86	0.25	8.31	1.96	26.32	0.11	43.76	0.16
T_m	0.15	0.02	0.27	0.02	0.65	0.03	0.99	0.03
SEL	4.01	0.25	8.58	1.95	26.98	0.12	44.75	0.18
T_d/SEL	96.3%	–	96.9%	–	97.6%	–	97.8%	–
T_m/SEL	3.7%	–	3.1%	–	2.4%	–	2.2%	–

(b) Ratio-method SEL

	500k		1M		3M		5M	
	Mean	σ	Mean	σ	Mean	σ	Mean	σ
T_d	0.99	0.16	1.17	0.21	1.40	0.30	1.63	0.41
T_m	0.06	0.01	0.10	0.02	0.32	0.02	0.50	0.02
SEL	1.05	0.16	1.27	0.22	1.72	0.31	2.13	0.41
T_d/SEL	94.3%	–	92.1%	–	81.4%	–	76.5%	–
T_m/SEL	5.7%	–	7.9%	–	18.6%	–	23.5%	–

(c) HEL

	500k	1M	3M	5M
HEL_0	1.4	1.6	2.0	2.4
HEL_1	4.4	10.1	28.9	47.2
HEL_2	2.4	3.0	3.7	4.5
improv.	45.5%	70.3%	87.2%	90.5%

VII. PERFORMANCE EVALUATION

In Cloud Shredder, only open and close operations consume extra time that users will notice. Because the performance degradation of closing a file can be inferred from the performance degradation of opening a file, we focus on the latency to open a file in Cloud Shredder. To evaluate this comprehensively and clearly, we define two different metrics:

System-Experienced Latency (SEL) is the time the system call takes to respond and open a file. SEL is the latency experienced by the upper layer of the system, such as user mode application. We insert codes in our wrapper of system call sys_open() to measure the downloading time (T_d), the merging time (T_m), and the SEL, when Cloud Shredder is activated. In our experiment, when Cloud Shredder is not activated, no matter how large the file is, SEL is almost a static number, about 0.2 ms, which is negligible. Therefore, we are just interested in the SEL when Cloud Shredder is activated, and the proportion of Td and Tm to SEL. In order to eliminate the random error of measurement, we ran a script to automatically open the same file ten times in a row and to calculate the mean value and the standard deviation (σ).

Human-Experienced Latency (HEL) is the time experienced by users when they open a file by graphical interface, such as when they double-click the mouse on the file icon. We manually measured by stopwatch three HEL values for the same file. HEL_0 represents the HEL of opening a file that is located outside the Safe Folder. Then we imported the file into the Safe Folder and use HEL_1 and HEL_2 to indicate the HEL of opening it using XOR- and Ratio-method respectively. Finally, we calculated $(HEL_1 - HEL_2)/HEL_1$ and got the performance improvement of Ratio-method.

We used a laptop with Intel Core2 Duo T9300 2.5GHz CPU and 3GB 800Hz DDR2 RAM, running Ubuntu 08.04 LTS Desktop Edition (kernel version: v2.6.21) and Amazon S3 as the cloud storage service. In order to mimic the travel scenario, we did the experiment in two different circumstances: in an office and at an airport. In each circumstance, we measured SELs and HELs of opening the same set of files ranging from 500kB to 5MB. We set the RR to 1/32 for Ratio-method.

In our office, we connected the laptop with LAN, whose actual internet access speed is about 1.5MB/s. As shown in Table II(a), using the XOR-method, T_d, T_m and SEL are roughly linear functions of file size. T_d/SEL and T_m/SEL are almost constant. From Table II(b), we find that, using Ratio-method, T_d, T_m and SEL are also roughly linear functions but with a smaller slope than when using XOR-method. T_d/SEL and T_d/SEL both change significantly. Comparing these two tables, we find that, when the file size is 500k, two method consume nearly same; however, when the file size is 5M, there is a great difference. Because in Ratio-method, the remote shares are much smaller than the ones in XOR-method, downloading them requires much less time. The larger the file is, the greater the difference. But, no matter whether the file is large or small, transmission overhead (by local network stack and storage cloud) is almost the same. As a result, when the file is small, the difference of Td and SEL is not significant. As shown in Table II(c), the Cloud Shredder with Ratio-method produces tiny extra HEL, and saves the HEL in the range of 10.5% to 53.6% compared with XOR-method.

In an airport, we connected the laptop with public Wi-Fi, whose actual internet access speed is about 175kB/s. As shown in Table II(a) and (b), T_d, T_m and SEL change in a larger range when using XOR-method, while the two proportion metrics fluctuate more when using Ratio-method. Both the two facts reflect the dominant influence of the time cost (network speed) to downloading the remote share. As shown in Table III(c), the Cloud Shredder with Ratio-method produces extra HEL of 2.1s, which should be acceptable for users. Compared with the XOR-method, Ratio-method saves the HEL in the range of 45.5% to 90.5%. We found the performance improvement of Ratio-method is more significant in the airport circumstance, especially for the relatively large files.

With the remote ratio set in our experiment, the time

overhead introduced by Cloud Shredder is acceptable for the files up to 5MB. For the larger files, a smaller remote ratio will not hurt the security but further reduce the time overhead. In practical application, an adaptive RR depending on the file size will be a better solution.

VIII. Discussions and Future Work

Why not use encryption? One could wonder why we do not encrypt the sensitive files by classical encryption techniques like AES and host the keys in the cloud. If so, the T_m will decrease significantly because of the small size of the key. However, it sacrifices the security at the same time. Similar to the FDE, it leaves full cipher-text on the local hard drive and put the data at potential risk under the attacker's computing power progress in the future. Once the attackers can compromise the encryption, they do not need to get the encryption key, no matter the key is stored in the USB device or in the cloud. (The FDE solution may also use key derivation scheme to postpone the compromise, but it does not have the time window to restrict how much time the attackers spend on attacking.) This is the common weak point of all encryption-based protection but our approach does not have.

Need extra authentication? One could propose an alternative design, in which an authentication server is built to authenticate users with passwords. The advantage is that no cypher text of SAK stored on the hard drive, but the side effect is that we need to maintain the relationship between users and their passwords and cloud credentials. Actually, the existence of SAK cypher-text does not weaken the security of Cloud Shredder, because it does not help at all for breaking into the cloud. Another related issue is how to authenticate the user when he/she "ask"(e.g., via phone call) the cloud service to carry out the reaction. We only present a minimal system in this paper, assuming this issue will be solved in practice.

Network failure: It is true that our approach's usability replies on the internet connection. However, as we have stated in the Section III.A that the travelers actually do not need to continuously access their confidential files during the trip. Nowadays public internet access service infrastructure is generally satisfactory to business travelers' needs. In the future the condition will go on improving. Therefore, as an approach designed for the cloud computing age, the limitation of internet availability should not be concerned too much. Of course, the possible sudden network disconnection should be dealt with. A special component will be added in the future and maturer version of Cloud Shredder to save and protect the files which have been modified but can not be uploaded temporarily.

Time overhead: The Ratio-method has improved Cloud Shredder's performance a lot, and the time overhead for confidential files of representative size carried in travel is acceptable. However, the time overhead may still influence the user's experience when dealing with larger files. The further optimization of the splitting method will help, but this is mainly limited by the current internet access speed. In the foreseeable future, with the upgrading of mobile and wireless networks, (e.g., 4G mobile network and FTTx internet access for WiFi will reduce the time overhead for one or two orders of magnitude) the Cloud Shredder will be used more smoothly on laptops, netbooks, tablet computers, and smart phones.

Optimization of the splitting algorithm: The splitting algorithm influences both performance and secure strength of Cloud Shredder system. There is a trade-off between these two sides. The further study to find the most-balanced algorithm in different circumstances is one of the valuable future tasks.

Transplanting to more platform: In this paper, we focus on the proposal of a novel data protection approach, which is independent from specific operating systems. Transplanting Cloud Shredder to other mainstream platform, such as Microsoft Windows and Mac OS, is basically an engineering task. We will work on it in the future.

IX. Conclusion

Cloud Shredder, without changing the user experience, provides enough security to avoid data disclosure when the laptop is stolen in travel.

Acknowledgment

Liu was supported by AFOSR FA9550-07-1-0527 (MURI), ARO W911NF-09-1-0525 (MURI), NSF CNS-0905131, NSF CNS-0916469, and the US-UK Army ITA program.

References

[1] CSI, "CSI/FBI Computer Crime and Security Survey, 2005–2009," http://gocsi.com, 2009.
[2] PRC, "Chronology of data breaches 2005–present," http://www.privacyrights.org, 2011.
[3] PI, "Ponemon institute, llc.: Airport insecurity: The case of lost & missing laptops," www.ponemon.org, 2006.
[4] Microsoft, "Windows bitlocker drive encryption frequently asked questions." http://technet.microsoft.com/en-us/library/cc766200(WS.10).aspx, 2009.
[5] N. Ferguson, "Aes-cbc + elephant diffuser: A disk encryption algorithm for windows vista. http://www.microsoft.com/downloads/details.aspx?familyid," 2006.
[6] P. Crowley, "Mercy: A fast large block cipher for disk sector encryption," 2000, pp. 49–63.
[7] K. Gjosteen, "Security notions for disk encryption," in *In Computer Security C ESORICS 2005*. Springer-Verlag, 2005, pp. 455–474.
[8] http://aws.amazon.com/s3/.
[9] https://www.dropbox.com/.
[10] http://rapidshare.com/.
[11] P. MacKenzie and M. K. Reiter, "Delegation of cryptographic servers for capture-resilient devices," in *CCS '01*. New York, NY, USA: ACM, 2001, pp. 10–19.
[12] F. Yao and Y. Yin, "Design and analysis of password-based key derivation functions," *Information Theory, IEEE Transactions on*, vol. 51, no. 9, pp. 3292 – 3297, sept. 2005.
[13] Y. Suga, "A low-cost key derivation scheme for hierarchy-based access," in *Network-Based Information Systems (NBiS), 2010 13th International Conference on*, sept. 2010, pp. 564 –568.
[14] http://wiki.debian.org/ramfs.
[15] DoD, "Department of defense: National industrial security program operating manual," 2006.
[16] B. Schneier, *Applied cryptography (2nd ed.): protocols, algorithms, and source code in C*. New York, NY, USA: John Wiley & Sons, Inc.
[17] K. e. a. Kurosawa, "Nonperfect secret sharing schemes and matroids," in *EUROCRYPT '93:*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1994, pp. 126–141.
[18] H. Krawczyk, "Secret sharing made short," in *CRYPTO '93*. London, UK: Springer-Verlag, 1994, pp. 136–146.
[19] C. Shannon, "Communication theory of secrecy system," 1949.
[20] http://en.wikipedia.org/wiki/Key_size.