A Zytek Communications Corporation White Paper

The Forgetful Disk Drive

Stephen Melvin, Ph.D.

September 9, 2010

The Forgetful Disk Drive

or

Cryptographic Data Deletion Using Time-Based Key Management

Abstract

Approaches to data protection have traditionally focused on physical security and/or cryptographic security, but the automatic deletion of data as a form of protection has not been widely employed. However, a vast amount of data is stored that has only a limited useful life and presents a risk to privacy and security after that time period. Approaches to data deletion can be broadly classified as physical destruction, data overwrite and cryptographic deletion. Cryptographic data deletion does not require any physical destruction and can be done quickly without any need to alter the data that is stored. Previous approaches to cryptographic data deletion have focused on bulk sanitization of storage devices. In this paper we introduce the notion of "forgetful" storage that employs cryptographic data deletion using a time-based approach to key management. By periodically erasing old keys and creating new keys, old data becomes automatically and instantly inaccessible, or virtually deleted. This data deletion can be accomplished without depending on any actions on the part of the user, application software or operating system. A refresh policy can be utilized to cause information that is accessed to be re-encrypted using a newer key, thereby extending the time before it will expire.

Introduction

Increasing amounts of personal, private, confidential or otherwise sensitive information is stored and replicated on storage devices. Traditional approaches to protecting such stored information can be classified as physical and cryptographic.

Physical protection involves ensuring that the hardware containing the stored information is physically secure and does not fall into the hands of those who might abuse it. Physical protection is not always practical, especially in the case of portable devices which are subject to loss or theft. Cryptographic protection involves encrypting stored information using one or more secret keys and protecting the security of the keys. Cryptographic protection has been widely employed on storage devices. Many disk drive manufacturers today offer full disk encryption and operating system support for disk encryption has been available for some time. For example, the Trusted Computing Group (TCG) publishes standards that specify host/storage device interaction for encrypted storage devices [1]. Also, drive encryption is available on some versions of Microsoft Windows [2].

Cryptographic protection relies on a secure key management system. If the key or keys are compromised, protection of the information may be lost. Some approaches to key management utilize specialized hardware to create and store encryption and decryption keys. For example, the Trusted Platform Module (TPM) is a specification that specifies among other things how keys can be managed securely [3].

Such approaches to data protection have focused primarily on cryptographic protection and secure key management. An independent but related concept is the issue of data deletion. In many cases it is desirable to specifically delete previously stored information. This can be in the case that a storage device is being decommissioned or discarded and/or the data is no longer needed or wanted. The issue of "sanitizing" disk drives before they are discarded has been considered by many and a number of approaches are available [4] [5].

The need to delete information is not just important for device sanitization. It may also be the case that the keys that protect data have been compromised so the data is no longer cryptographically secure. Also, there may be situations where the mere existence of the data, even if cryptographically protected, presents a liability. For example, otherwise cryptographically secure data may still be subject to a subpoena, such that it must be provided if appropriately requested. Truly deleted data is no longer accessible to anyone.

Approaches to data deletion can be broadly classified as physical destruction, data overwrite and cryptographic. Figure 1 illustrates a data protection taxonomy including both cryptographic protection and data deletion.

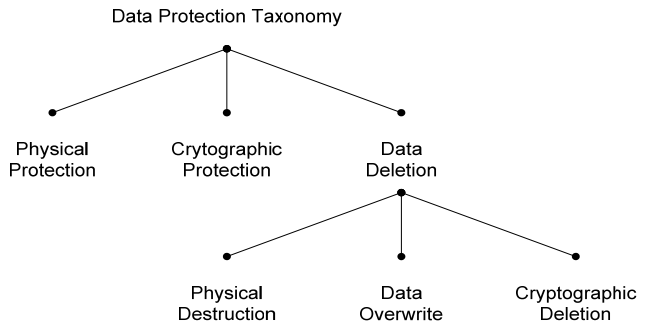

Figure 1 – Data Protection Taxonomy

In the case of physical destruction, the hardware containing the information is physically destroyed, rendering access to it impossible. Physical destruction of storage devices is cumbersome and may be unreliable. In the case of data overwrite, new data is written over previously existing data. Simple forms of data overwrite are also unreliable, potentially allowing recovery of overwritten data using specialized circuitry [6]. More conservative approaches to data overwrite, such as government sanctioned techniques involving multiple overwrites with different patterns can be very time consuming [4] [5]. We include in the category of data overwrite also the situation where power is removed from a volatile storage element, which is effective in certain situations but is not an available solution for nonvolatile storage, such as magnetic and EEPROM based storage devices.

Instant full drive sanitation using cryptographic data deletion is available in certain drives [7]. In this case data on the entire drive is rapidly made inaccessible by using an internal Fast Secure Erase command that deletes internal cryptographic keys. Unfortunately such an approach cannot be used incrementally to delete only certain data and depends upon the proprietary implementation of key storage on the device, so it is hard to verify. An approach to incremental deletion of stored information using "erasable memories" was explored in [8].

In this white paper we first discuss the implementation of cryptographic data deletion generally, which does not require any physical destruction and can be done without having to alter the stored data. Then we will discuss a disk drive system that allows cryptographic keys to be dynamically managed. Finally we will illustrate how time-based key management can be used to implement a storage device that will automatically expire old data.

Cryptographic Data Deletion

Cryptographic data deletion, like cryptographic data protection, involves encrypting information that is stored and decrypting it when it is retrieved. Such encryption can be based on a symmetric or non-symmetric system. In the case of a symmetric system, the same key is utilized for both encryption and decryption. In the case of a non-symmetric system different keys are utilized for encryption and decryption. Typically, non-symmetric encryption systems involve generating encryption and decryption keys in matched pairs and are designed such that knowledge of one key does not permit a practical discovery of the other key.

The principle behind cryptographic data deletion is to delete the decryption key or keys in order to accomplish a virtual deletion of the stored information. The deletion of the decryption key could itself be accomplished by physical destruction of a storage element, overwriting a storage element, or by cryptographic data deletion (if the decryption keys are protected by other keys). Cryptographic data deletion can result in a great increase in efficiency since the storage of the decryption keys may be many orders of magnitude smaller than the storage of the encrypted information itself. A cryptographic approach to data deletion can thus be done quickly without any need to change the data that is being deleted.

Key management and in particular the provision for key deletion is an important component of a cryptographic data protection scheme. However, many existing approaches to cryptographic data protection do not provide efficient and convenient techniques for key deletion. In the next section we will illustrate some examples of automatic key deletion implementations.

Dynamic Key Management

Figure 2 illustrates a dynamic key management mechanism interposed between a host device and a storage device. The storage device can be a conventional hard disk drive, a solid state drive or another form of digital storage. One peripheral interface interfaces with the host device and the other peripheral interface interfaces with the storage device. Generally speaking, the host device is a producer and/or consumer of information and the storage device is a repository of information. Read and write commands are received by the host peripheral interface and are communicated to the storage device peripheral interface. Similarly, responses to read and write commands are received by the storage device peripheral interface and are communicated to the host peripheral interface.

Typically, read and write commands operate on blocks of data. In the case of conventional hard disk drives, many drives in current use today utilize 512 byte sectors as the smallest unit of information that can be read or written. Read and write

commands received by the host peripheral interface in this case would consist of operations on integer multiples of 512 bytes. Other block sizes are possible, for example some optical disk drives utilize 2048 byte sectors and there have also been proposals for larger sector sizes (e.g. 4096 bytes) in hard disk drives.

When the host peripheral interface receives a write command from the host device, the sector or sectors being written are encrypted and a write command containing the encrypted sector or sectors will be passed to the data storage peripheral interface. The key or keys used to encrypt the sector or sectors are supplied by the key storage. The storage device will then receive a write command and will write the encrypted data to the specified sector or sectors.

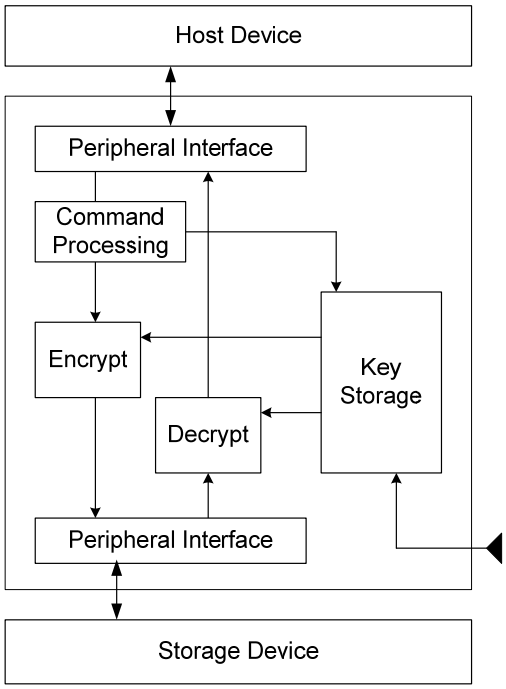

Figure 2 – Dynamic Key Management Overview

In the case of a read command received by the host peripheral interface from the host device, the command will be passed along to the storage device through the storage device peripheral interface. When the data for the sector or sectors is returned, the data will be decrypted and the decrypted data will be returned to the host device. The key storage will supply the key or keys to decrypt the data. The encryption and decryption are operated so as to guarantee that the encrypted data is returned to its original form.

The device illustrated in Figure 2 may be implemented such that it mirrors read and write commands across its peripheral interfaces and implements identical interfaces between the host device and the storage device. This allows the mechanism to be retrofitted into existing computer systems without any changes to existing hardware or software. The host device can communicate with the device in the same way as it would communicate directly with the storage device. Similarly, the storage device can respond to commands from the device in the same way as it would respond to commands directly from a host device.

With a non-symmetric encryption mechanism it would be possible to have the key storage store only the decryption key or keys and move the encryption apparatus into the host device. In that case the host device could have the encryption key or keys and could encrypt the information before sending it to the device.

The encryption / decryption mechanism and the length of the key or keys used are chosen to safely protect the data such that an inspection of the encrypted data makes recovering the original data impractical. Higher levels of protection typically require more computation on the part of the encryption and decryption circuits. The computational requirements can vary depending on the level of protection desired. In some cases, the data location, such as the sector number, can be used in the process of generating a key or keys for encryption and decryption. This guarantees that the same data will be encrypted differently if stored in different locations of the disk drive. The device could include hardware based on the Trusted Platform Module (TPM), which can be used to securely store keys and can be used to generate keys and key pairs using a built in random number generator [3].

In some implementations, a number of different keys are used to encrypt and decrypt information. For example, there could be a variety of different categories or groups of protected information, and they could each be encrypted with different keys. It may also be possible to selectively erase keys associated with certain categories of information and not to erase keys associated with other categories of information. In some implementations certain information is not encrypted and therefore is not protected.

A number of different key management mechanisms are possible in conjunction with Figure 2. In one case, encryption and decryption keys are generated internally and not accessible to the outside world. In this case the elimination of the keys achieves complete inaccessibility. In other cases, it may be desirable for a trusted system to generate keys and for those keys to be subsequently installed. In this case, if the key is locally eliminated, the stored information can still be made accessible by going back to the original key generator. This may be a viable solution for portable devices since deletion of the decryption key will cause the information to be locally inaccessible, but may still be recovered using information stored in a central (and presumably more secure) location.

The command processing apparatus illustrated in Figure 2 is used to allow dynamic key management through applications running on a host system, such as one coupled to the host peripheral interface. Dynamic key management refers to the ability to perform operations on stored keys while the system is in operation. In one example, peripheral commands are examined by the command processing circuit and those related to key management are intercepted and processed. Key management operations include key creation, key storage, key retrieval and key deletion. Key management operations can also be accomplished using an external port. For example, the external port could be a USB port that allows an external host system to perform key operations. In this way, key management can be controlled internally (using the main information data path) and/or externally (using an external port).

In the case of applications that utilize the command processing apparatus, these applications can be firmware running on the host device, BIOS software, OS drivers, OS daemons, user applications or some combination of the above. In some cases the operations on the key storage can be effected in a way that is transparent to the storage device using regular disk drive read and write commands with special data patterns. In other cases new or reserved commands can be utilized to communicate between the host device and the command processing circuit.

Time-Based Key Management

Figure 3 illustrates aspects of a time-based key management system. A time-based key management system will typically periodically create new keys and delete old keys based on a schedule. For example, a time based key management system could create a new key and delete the oldest key once per day. Such a system allows information older than a predetermined number of days to be virtually deleted, or to expire. For example, if 731 keys are stored by the key storage, one per day, then information that was encrypted using a key more than two years old will be automatically forgotten.

The data status table records for each block of data on the storage device the identification of a key used to encrypt the stored information. The block of data could be a sector number or could be some larger group of data, such as a file system cluster. The data location, such as the sector number, is used to index into the data status table. In one example 731 keys are stored in the key storage and the data status table stores a 10-bit index indicating which of the 731 keys should be used to decrypt the information currently stored at the location in question. The real time clock circuit is used to keep track of the current date and time, and to allow the periodic creation of new keys and erasure of old keys.

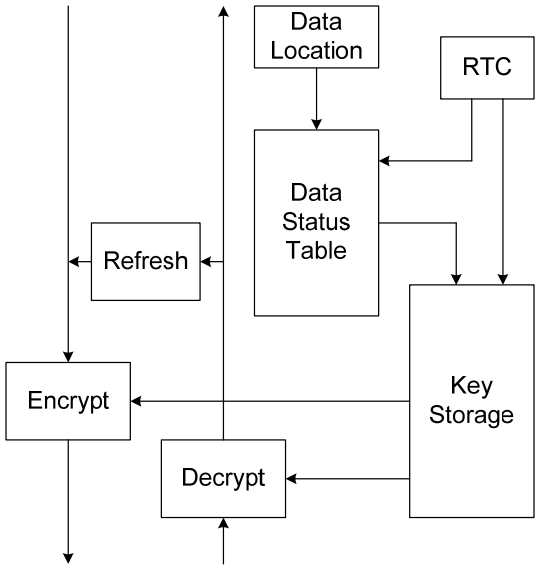

Figure 3 – Time-Based Key Management

Figure 4 illustrates an example Key Storage Unit that could be used in a time based key management system. For example, every day at 12:00 AM a new key could be created and data written to the key storage. The oldest key would also be erased, for example the key associated with the same date two years previous. In the case of data being written to the storage device, the data status table is able to retrieve the current key of the day and update the entry for a specific data location. This will automatically give newly written data a two year life span.

In some situations, it may be desirable to automatically refresh data when it is being read from the storage device. In this case, it is necessary to decrypt and then re-encrypt the information using a different key. Data from the data decryption circuit is passed through the refresh path to the encryption circuit and re-written to the same location. It may be desirable to use the current *key du jour* when a refresh operation is performed. Such a policy would mean that data would expire two years from the initial write or the most recent read, whichever is later. Other refresh policies are possible. For example, it would be possible to split the difference between the date associated with the data being read and the current date. This policy increases the life of the data being read, but not to a full two years. Another example is to decrement the key index by a certain number of days each time it is accessed. Thus, if the information is accessed frequently, it will have a longer life than if it is accessed infrequently. Such a policy in some ways mimics human memory recall in which information recalled more frequently

and information that is more recent has a better chance of being remembered accurately.

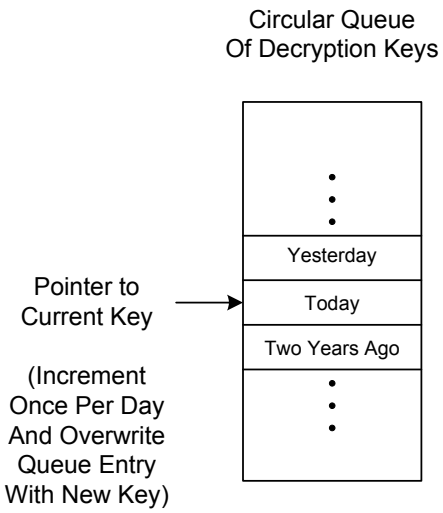

Figure 4 – Key Storage Unit

In other situations it may be desirable to expose the expiration date of the data to the operating system and allow the OS to choose the expiration date within the window of available time periods (e.g. a two year window). For example, the date of access or the date of modification of a file could be used as the appropriate date for encrypting data being read or written. In this way new data being written (if had not been newly created) could use an older key rather than the current key of the day. This allows the OS to specify a specific expiration date of data being written to the disk drive. It would also be possible to use special codes in the data status table to mark unencrypted data or expired data. It is also possible to have different refresh modes depending on the location of the data, depending on instructions from the host, or as a device specific preference.

Another way the operating system could manage data expiration would be to specify a "time to live" of the data by counting backwards from the oldest key. For example, if one key is generated and deleted per day, then by specifying the seventh oldest key in the queue, the data will automatically expire in one week.

In some implementations caches are used within the device or within the storage device. These caches, which store data being read from or written to the storage device, can implement a write through or a write back policy. A cache could be employed to store data in its unencrypted state or its encrypted state or both. In the case of a write-through cache the actual write of modified data to the storage device is deferred. A cache may allow the device to defer the re-encryption of refreshed data until such time as it is necessary to flush the modified data to the storage device.

A series of steps are invoked periodically, for example at 12:00 AM as an interrupt generated by a real time clock circuit. First, the oldest key is erased. By deleting the key associated with a certain time period, all data that was encrypted associated with that time period will be virtually deleted. Then a new key is generated. This can be done according to known methods in a variety of different ways. In some implementations a hardware random number generator is used to generate a new key or a new key pair in a secure environment. Finally, the key table is updated to reflect a new "current" key, which can be used to encrypt newly written and refreshed data.

Conclusions

In this paper we have discussed how cryptographic data deletion and a time-based key management scheme can implement the automatic expiration of stored data. A so-called "forgetful" disk drive will automatically forget old data without any actions on the part of the user, operating system or application. While permanent archival storage is not a target application for such a product, there are many scenarios where such a device would be useful and practical. Some have argued that unlimited memory is a threat to society and culture and that data expiration technology should be incorporated into devices [9]. In any case, it is clear that there is a vast amount of data that has only a limited useful life and presents a risk to privacy and security after that time period. By recording such data on a forgetful disk drive, one can be ensured that old data is no longer available.

References

1. TCG Storage Security Subsystem Class: Opal Specification Version 1.00, 4 February, 2010, TCG Storage Architecture Core Specification, Version 2.00, 20 April, 2009.

2. "Encrypting File System," Wikipedia, http://en.wikipedia.org/wiki/Encrypting_File_System.

3. TPM Main Part 1 Design Principles Specification Version 1.2, 9 July 2007, TPM Main Part 2 TPM Structures Specification version 1.2, 26 October 2006, TPM Main Part 3 Commands Specification Version 1.2, 26 October 2006.

4. S. L. Garfinkel and B. Shelat, "Remembrance of Data Passed: A Study of Disk Sanitization Practices," IEEE Security & Privacy, January / February 2003, pp. 17-27.

5. D. M. Commins, T. Coughlin and G. F. Hughes, "Disposal of Disk and Tape Data by Secure Sanitization," IEEE Security & Privacy, July / August 2009, pp. 29-34.

6. P. Gutmann, "Secure Deletion of Data from Magnetic and Solid-State Memory," Proc. Sixth Usenix Security Symposium, Usenix Association, 1996.

7. G. Hughes, "Wise Drives," IEEE Spectrum, Aug. 2002, pp. 37–41.

8. G. Di Crescenzo et al., "How to Forget a Secret," Symposium on Theoretical Aspects in Computer Science (STACS 99), Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1999, pp. 500–509.

9. V. Mayer-Schönberger, Delete: The Virtue of Forgetting in the Digital Age, Princeton University Press, 2009.