Architecture for a Non-Copyable Disk (NCdisk) Using a Secret-Protection (SP) SoC Solution

Michael S. Wang and Ruby B. Lee
Department of Electrical Engineering, Princeton University, Princeton, NJ 08544 USA

Abstract – **Piracy of copyrighted digital contents, such as movies and music is rampant in cyberspace. A piece of digital material may be repeatedly copied and proliferated throughout the Internet with ease. We examined both software and hardware vulnerabilities in existing digital copy-protection methods. As a result, we propose a non-copyable disk (NCdisk) that makes it significantly harder for digital contents to be copied. Any digital content written onto the NCdisk can only be read through a predefined set of outputs of the NCdisk, and the original plaintext digital form may never be read out of the NCdisk. We add a minimal set of components based on the Secret-Protection (SP) architecture to the existing disk's SoC chipset to attribute the disk with the non-copyable property. We further present the security protocol to be used along with the NCdisk to provide a copy-protected digital movie download scenario.**

I. INTRODUCTION

Today, an immense amount of information exists in digital form. A large percentage of it is copyrighted contents that should only be available to authorized users. In such cases, the user is usually permitted to read (or play) the contents but should not be allowed to copy and distribute the contents. Nevertheless, unauthorized copying and distribution of digital contents occur frequently and is a major problem for many content providers.

This content-piracy problem is currently a serious concern for the movie and music industry. Due to the increasing Internet bandwidth and the emergence of more powerful portable player devices, the demand for directly downloading media contents from the Internet to an end-user's player device is on the rise. A typical copy-protection method [1] used to prevent the illegal copying of these media contents is as follows: a content provider installs his own software onto the user's player device, such as a PC, an iPod, etc. Then, the provider sends encrypted contents to the user's device. In order to obtain the keys used to decrypt and read the encrypted contents, the user must authenticate with the content provider or with a third party licensing clearinghouse. Next, the keys are sent to and hidden on the user's device. Only the content provider's installed software on that device can find and use the keys to decrypt the encrypted contents. Hence, this copy-protection method restricts the copying of contents by sending only encrypted contents over public networks, hiding keys on the user's devices, and allowing only the provider's special software to find and use these keys.

A major weakness with the existing copy-protection method described above is that the encrypted contents are sent to various kinds of player devices that do not have secure processing architectures to hide the decryption keys. In the underlying processor architectures, machine instructions, registers, memories and buses are open resources that can be controlled or accessed by the operating system (OS), application software and also by malicious software. Furthermore, since both the application software and the OS can have bugs and software vulnerabilities, hackers can use these software weaknesses to find the hidden decryption keys.

We propose a non-copyable disk (NCdisk), which is a storage device that automatically encrypts all data written into it and does not allow the plaintext form of the data to leave it except through controlled display outputs. We propose a minimal set of changes to an existing disk controller System-on-Chip (SoC) to attribute the disk with the non-copyable property. Our proposal is based on the Secret-Protection (SP) secure processor architecture [2][3][4], which provides a secure environment to store critical secrets and allows only a trusted software module to access these critical secrets.

II. THREAT MODEL

We assume that the content provider can write a trusted software module that will be allowed to use and access critical secrets but cannot leak these secrets out. Further, we assume that any other software is un-trusted and should not be allowed to access critical secrets. The attacker is able to mount software attacks. He can monitor network transactions and probe external memories and buses. We assume that physically probing inside a chip, such as a System-on-Chip (SOC) is more difficult without destroying functionality, and hence is not in our threat model. We also do not consider side-channel attacks on a SoC

III. NCDISK CONCEPT

Figure 1 shows a flowchart of the NCdisk concept. The NCdisk is a data storage device, in which any digital content written into the device is automatically encrypted using a key that is generated by the NCisk that never leaves the NCdisk. All data stored on the NCdisk are in such an encrypted form, and the stored data can only be read through a set of predefined outputs, such that the digital plaintext form of the data never leaves the NCdisk.

Both plaintext data and encrypted data may be written onto the NCdisk. Each encrypted data is encrypted using a

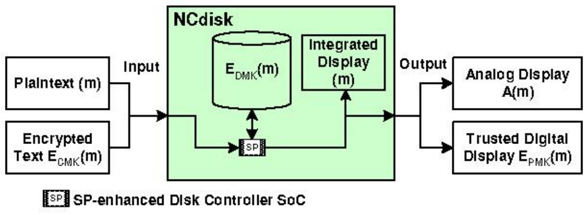
NCdisk
Plaintext (m)
Encrypted Text $E_{CMK}(m)$
Input
$E_{DMK}(m)$
Integrated Display (m)
SP
Output
Analog Display A(m)
Trusted Digital Display $E_{PMK}(m)$
SP-enhanced Disk Controller SoC

Fig 1. NCdisk Concept

secret key, called the Content-provider Media Key (CMK), which is known only by the content provider and the NCdisk. The CMK is never actually stored anywhere but is instead generated using a shared key between the content provider and the NCdisk. We examine the detailed key management protocol in Section V. Both the content provider and the NCdisk must have a secure location to generate and use the CMK so that it is not revealed to anyone else. We assume that the content provider has such a secure location, and we show in Section IV how the NCdisk achieves this. If the CMK is kept secret, then the plaintext form of the encrypted data will not be leaked out during the input phase of the NCdisk.

Either plaintext data or CMK-encrypted data can be input into the NCdisk. In the former case, it is first encrypted using the Device Media Key (DMK). In the latter case, the NCdisk first decrypts the data using the CMK and then re-encrypts the data using the DMK. The DMK is generated within the NCdisk and it never leaves the disk. We discuss in Section IV how to keep the DMK secret from everyone, including the user of the disk. Note also that each input data to the NCdisk is encrypted using a different DMK, as described in detail in Section V. Encrypting all the data stored on the NCdisk using the DMK protects the storage phase of the data, by ensuring that the plaintext version of the digital data never resides on the disk.

Any data stored on the NCdisk can only be read out of the disk through a pre-defined set of output channels. An encrypted digital data can be decrypted and converted to an analog format, which can then be sent out of the NCdisk. Alternatively, an encrypted digital data can be decrypted using the DMK and re-encrypted using a Player Media Key (PMK), which is only known by a trusted digital display and the NCdisk. Both the trusted digital display and the NCdisk must have a secure location to generate and use the PMK so that it is not revealed to anyone else. The PMK-encrypted data is sent out of the NCdisk. Third, if the NCdisk has an integrated display, such as a built-in LCD screen like in the iPod devices, then the NCdisk may decrypt the stored data and send the digital streaming data to the integrated display. It is assumed that it is hard for a casual attacker to siphon off information on the internal link connecting the NCdisk and its integrated display. Note that this integrated display is not foolproof against more dedicated attackers. Nevertheless, this integrated display raises the bar against possible attacks to siphon off information. In all three pre-defined output channels, the (high-quality) plaintext version of the digital data never leaves the NCdisk in the output phase of the NCdisk. To summarize, the NCdisk ensures that no one, not even the legitimate user of the NCdisk, can obtain a copy of the digital plaintext version of the data stored on the disk.

The NCdisk addresses some of the weaknesses of existing copy-protection methods. Instead of sending copyrighted movie or music contents to insecure PCs or portable media players, a content provider can instead send these contents to a user's NCdisk. In a way, the NCdisk functions like a book in that only those people who have physical possession of the NCdisk can view the contents stored on it. Just as it would be very inconvenient for a person to copy and distribute the book, a user would have a very difficult time trying to copy the original digital plaintext data stored on the NCdisk. However, unlike a book, the NCdisk provides the convenience of directly downloading and viewing copyrighted digital contents without the need to physically travel to a store. Also, an NCdisk can store many items of digital content.

IV. NCDISK SP-BASED SOC ARCHITECTURE

The NCdisk concept ultimately boils down to achieving two goals. The first goal is to enable the NCdisk to be able to store secret keys and ensure that these keys never leak out of the NCdisk. The second goal is to fully predefine how data can be read out of the NCdisk such that the original digital plaintext data is never leaked out. We do not achieve these two goals by redesigning a completely new disk architecture from scratch. Instead, we only need to be concerned with the disk controller components, which control how data is written in or read out of a disk. We achieve these two goals by implementing a SoC consisting of existing disk controller components, plus a minimal set of additions. This new SoC can then be connected to the rest of the existing disk components to turn an existing disk [5] into an NCdisk (see Figure 2.)

The existing disk controller components in the SoC include a disk controller processor, a read/write buffer

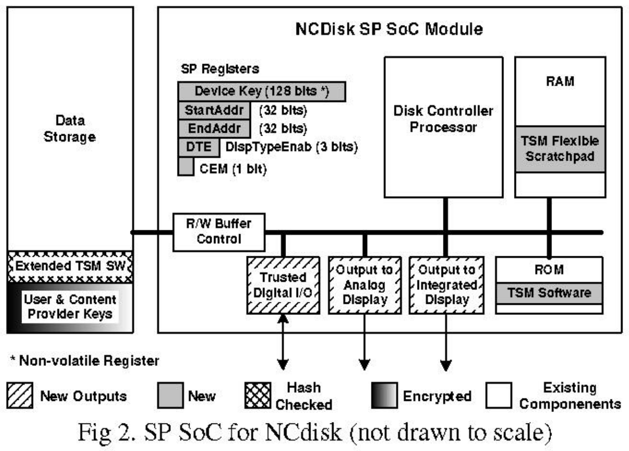

Fig 2. SP SoC for NCdisk (not drawn to scale)

control, and some RAM and ROM memory. The additions are divided into two types. The first type of additions comes from the Secret-Protection (SP) architecture[2][3][4], which provides a secure environment for a set of trusted software to access critical secrets, while preventing these secrets from leaking out of the SoC. The SP additions include new SP registers and hardware support for new SP instructions. Also, portions of the RAM and ROM are dedicated for SP software. The second type of additions is the output interface. There are three different output interfaces, which encompass the predefined set of outputs. Below, we examine how this SP SoC achieves the two goals of the NCdisk.

A. Storing and Protecting Keys

The SP-based SoC stores keys in two places. First, the SoC stores a 128-bit key called the device key in a non-volatile on-chip register. All NCdisks are manufactured with an empty device key register, and after deployment, each NCdisk self-generates its own device key. This device key will never leave the NCdisk. Second, user and content provider keys can all be stored off-chip, encrypted with the device key. Since the protection of the off-chip keys hinge on the protection of the device key, we focus on efforts to protect this on-chip device key.

The Trusted Software Module (TSM) of the SP architecture plays an important role in protecting the device key. TSM ensures that no software can access the device key register. Instead, only the TSM software stored in the on-chip ROM can get a key that is derived from the device key. Table 1 contains SP instructions used for protecting the TSM and its execution.

SP Instruction	Description
Begin_TSM (on-chip ROM)	Begins execution of the TSM (First enables access of TSM scratchpad memory)
End_TSM (TSM only)	Ends execution of the TSM (First disables access of TSM scratchpad memory)
SecureMem_Set (TSM only)	Sets the StartAddr and EndAddr registers to define the TSM scratchpad memory
DK_Derive_Key (TSM only)	Derive a new encryption key using the device key and an input key id

Table 1. SP Instructions for NCdisk

The SP instruction Begin_TSM turns on the Concealed Execution Mode (CEM) status bit register, while the End_TSM instruction turns off the CEM status bit register. The Begin_TSM instruction can only be invoked by programs stored on the on-chip TSM ROM. Also, the other SP instructions, such as End_TSM, SecureMem_Set, and DK_Derive_Key may only be invoked when the CEM status bit is turned on. This implies that the TSM software stored in ROM can execute the SP instructions only after the Begin_TSM instruction is invoked first. Further, this TSM ROM code can call more complicated TSM code that is stored off-chip. This extended TSM code must be integrity-checked before it is run. Note that since all TSM software must start off with the Begin_TSM instructions, which can only be invoked from the ROM, no other external software can call the DK_Derive_Key instruction to derive an encryption key using the device key. The DK_Derive_Key is the only instruction that can use the device key. Even this instruction cannot read out the value of the device key to register or memory. Instead, it can only use the device key to derive different encryption keys for different files stored on the NCdisk. Finally, to ensure that no non-TSM software may run during CEM, we disable interrupts during CEM mode.

Simply disallowing non-TSM software to access the device key register is not enough to prevent the register content from leaking out of the SoC. The run-time data generated by the TSM software in ROM must not be leaked out of the SoC because this data may include information that can reveal the device key. Similar to the sensor-mode SP architecture [4], we propose to dedicate a portion of RAM as TSM scratchpad memory. This scratchpad memory can only be accessed when the CEM status bit is on. We propose a new SP instruction called SecureMem_Set, which can set the start and end addresses of the scratchpad memory. The start address and end address are stored in the new 32-bit StartAddr and EndAddr registers. The SecureMem_Set instruction can change the values of these registers. When the SoC is in the CEM mode, the memory location between the StartAddr and EndAddr becomes accessible to the TSM software, which is the only software that can run during the CEM mode. However, when the SoC is not in the CEM mode, this scratchpad memory will not be accessible by any instructions. The purpose of having a flexible TSM scratchpad memory is to give the TSM software programmer the ability to decide how to allocate the RAM between TSM trusted access and general access areas.

Further, not allowing any software to directly access the device key and preventing the run-time data of the TSM software from leaking out of the SoC still does not ensure that the device key will not leak out of the SoC. The TSM software must be carefully written to ensure that this trusted software does not send any infomation that can be used to detect bits of this key, out of the SoC. The TSM software for the NCdisk contains a fixed set of API functions (see Table 2). An external control can only use the NCdisk by calling one of these predefined functions. None of these API functions will output the device key from the SoC.

API Function	Description
TSM_Write	Write data into NCdisk
TSM_Read_Analog	Output to analog channel
TSM_Read_Trusted	Output to trusted display
TSM_Read_Integrated	Output to integrated display

Table 2. TSM API

B. Controlled Predefined Output

The second goal of the NCdisk is to predefine how data can be read out of the NCdisk such that the original digital plaintext data is never leaked out. This goal is achieved using the TSM API functions. There are three API functions for reading data out. Each API function reads data out through a different output interface on the SoC. This goal is achieved since these three API functions provide the only way for external control to read data out of the NCdisk and none of these API functions will leak out the original digital plaintext data.

The TSM_Read_Analog function decrypts the stored data, converts it into an analog format through the D/A converter, and sends it out of the NCdisk. This API function performs the analog conversion immediately after the decryption, and since interrupts are disabled, the plaintext *digital* data (e.g., high fidelity movie) will not leak out.

The TSM_Read_Trusted function decrypts the stored data with the DMK, re-encrypts it with the PMK, and sends encrypted digital data out of the NCDisk. Each data has its own DMK, and each trusted display may have its own PMK. The DMKs, CMKs, and PMKs are never stored anywhere. Instead, they are deleted right after encryption and re-derived upon decryption by a shared key or device key by the trusted software.

Finally, the TSM_Read_Integrated function decrypts the stored data and sends it to an integrated display through the integrated display interface. Since the display is integrated with the NCdisk, we assume that it is much harder for an attacker to siphon off the data on the internal link connecting the integrated display to the NCdisk.

V. NCDISK SECURITY PROTOCOL

Before presenting the NCdisk security protocol, we first examine an online movie download scenario for using the NCdisk, as shown in Figure 3.

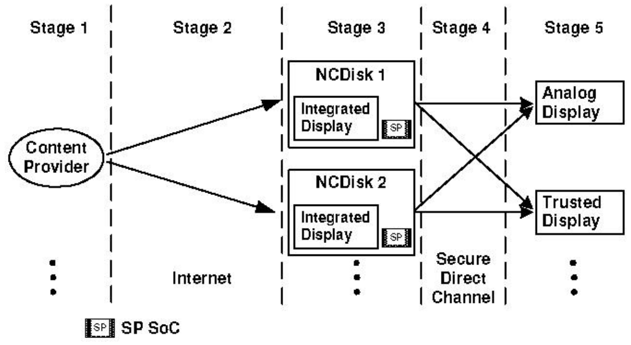

Fig 3. Online Movie Download Scenario using NCdisk

At stage 1, a content provider has a database of movies. The content provider sends the movies in encrypted form to its users through the public Internet (insecure) at stage 2. At stage 3, the users receive this encrypted movie. In existing DRM systems, users store these encrypted movies, along with the decryption keys on their PC or portable media players, which typically do not provide adequate protection for these keys. As a result, these movies may be easily copied and proliferated through the Internet. However, if the users store these movies onto their NCdisk, then those movies will not be easily copied in their original high-quality digital plaintext form. Instead, the movies can only be viewed through the NCdisk and one of three pre-defined type of outputs. The NCdisk can provide both security for the content providers and convenience for the users. The NCdisk can be used to replace the current DVD-by-mail rental services such as Netflix™ and Blockbuster™. These service providers currently mail millions of DVDs to all their users each week. Instead, these service providers could mail to each user a single NCdisk. Using this NCdisk, the users can download their desired movies without having to wait for DVDs by mail, and the content providers will have the assurance that their movies cannot be mass copied in their original plaintext digital form.

We present a security protocol to use along with the NCdisk for an online movie download application, as shown in Table 3. When the NCdisk is manufactured, it is completely empty. It has no movies stored on it, and its SoC registers and memory are void of keys and software. The manufacturer ships these blank NCdisks to the movie content provider, who is the trusted party in our model. The 5 paragraphs below refer to parts (a) through (e) in Table 3.

The content provider installs an initialization software on the NCdisk that will self generate a random device key and store it in the device key register, load up the trusted TSM software, write in a unique serial number (SN) identifying the NCdisk, and load up the shared key that the content provider shares with the NCdisk. Afterwards, the content provider can remove the initialization software. Note that since the device key and TSM software are in ROM (write-once Flash) memory, no one can re-program the NCdisk in the future. This prevents an attacker from tampering with the NCdisk. After the NCdisk is initialized, it can be deployed to users.

When a user connects the NCdisk Dj to the content provider through the Internet, the content provider can read off the SN on Dj. The content provider has a database that associates each SN with the corresponding key that it shares with that Dj. Using this shared key, the content provider can securely send movie contents to Dj.

Further, the content provider builds a movie database, where each movie M^i is encrypted using a different movie encryption key CMK^i. These movies and their keys are assumed to be stored in a secure location on the content provider's server.

A particular movie M^i is sent to numerous NCdisks. The content provider prepares a different bundle for each NCdisk Dj. The bundle consists of three components. The first component is the encrypted movie, which is encrypted with the movie encryption key CMK^i. The second component is the encrypted CMK^i. For different Dj, the CMK^i is encrypted using a different key K^i_j. This key is always derived upon use and immediately deleted afterwards. Only the content provider and the NCdisk can re-derive K^i_j. The third component is a random value MID^i_j that identifies that particular movie M^i and that particular NCdisk Dj. This value plays a role in re-deriving K^i_j, but it is only useful to the content provider and the NCdisk.

When an NCdisk receives a movie bundle, it re-derives K^i_j and uses K^i_j to obtain the original movie encryption key CMK^i, which can be used to obtain the plaintext movie M^i. Next, the NCdisk re-encrypts M^i using a new movie encryption key DMK^i_j, which is derived using the NCdisk's unique device key, Dj. At this point, no one can copy the original digital plaintext movie from the NCdisk. Using the predefined API functions, the NCdisk provides a set of controlled outputs to ensure that the original digital plaintext movie does not leak out during the output phase.

C: content provider, **D**: NCdisk, **SN**: serial number, **M**: movie content; **i**: ith movie, **j**: jth NCdisk.

Manufacturer provides a blank Dj
Manufacturer builds a blank NCdisk Dj that does not have any software or keys stored inside.

(a) Manufacturer sends Dj to C for initialization
1. C loads secure installation SW into NCdisk RAM.
2. The initialization SW generates a random device key DKj and writes DKj into the device key register, which is a non-volatile register (or write-once Flash memory).
3. The initialization SW loads TSM SW into the write-once memory area. Further, the initialization SW generates a keyed hash of the extended TSM SW and stores this extended TSM SW and its keyed hash in the off-chip data storage area.
4. The initialization SW stores a unique SN into the write-once memory area.
5. The initialization SW generates a random key CDKj, which it shares with Dj. It encrypts CDKj using the device key and stores it in the off-chip storage area.
6. Finally, C removes the initialization SW, disables the writing of the on-chip Flash memory, and the NCdisk is fully initialized.

(b) C distributes Dj to user j
1. User j buys Dj from a store, or C sends Dj to user j.
2. User j connects Dj online to C's website
3. C reads SN from Dj. C has a database that associates each SN with a CDKj, which C shares with Dj. Using CDKj, C securely sends data to Dj.

(c) C builds a movie database
1. C generates a random movie encryption key CMK^i for each movie M^i.
2. C encrypts movie M^i with CMK^i.
3. C saves $E_{CMK^i}(M^i)$ and CMK^i in movie database.
4. C periodically re-encrypts M^i with a new CMK^i

(d) C prepares M^i to send to Dj
1. For a given M^i, C prepares a different M^i bundle for each Dj as follows:
   a. C searches up the CDKj that it shares with Dj
   b. C generates a MID^i_j identifying M^i and Dj
   c. C derives a key $K^i_j = MAC_{CDKj}(MID^i_j)$
   d. C encrypts CMK^i with K^i_j
   e. C sends the bundle for M^i to Dj:
      $\{E_{CMK^i}(M^i), E_{K^i_j}(CMK^i), MID^i_j\}$

(e) Dj processes bundle before storing it
1. Dj first decrypts the bundle to obtain plaintext M^i:
   a. Dj re-derives $K^i_j = MAC_{CDKj}(MID^i_j)$
   b. Dj decrypts $CMK^i = D_{K^i_j}(E_{K^i_j}(CMK^i))$
   c. Dj decrypts $M^i = D_{CMK^i}(E_{CMK^i}(M^i))$
2. Dj then re-encrypts M^i for storage
   a. Dj generates a random ID^i_j identifying M^i and Dj
   b. Dj uses device key DKj and ID^i_j to derive a new movie encryption key:
      $DMK^i_j = MAC_{DKj}(ID^i_j)$
   c. Dj encrypts M^i with DMK^i_j
   d. Dj throws away DMK^i_j
   e. Dj stores M^i bundle, which is now non-copyable
      $\{E_{DMK^i_j}(M^i), ID^i_j\}$

Table 3. NCdisk Security Protocol

VI. CONCLUSION AND FUTURE WORK

We proposed a novel Non-Copyable, NCdisk, concept to prevent copying of the digital content stored in a disk. The NCdisk concept boils down to implementing two design goals: protecting secrets and providing output control. We achieve these two goals by implementing a Secret Protection (SP-based) SoC architecture that can be added to existing disk architecture to turn that disk into an NCdisk. Further, we design a security protocol that can be used along with the NCdisk to provide security and convenience for the online movie download application. This paper also illustrates the use of SP architecture in novel applications such as NCdisk. A future goal of this project is to extend the NCdisk architecture and security protocol to support multiple content providers. Future research also includes extending the NCdisk to other applications besides online movie download, using the NCdisk to provide greater copy-protection and privacy-protection for sensitive data.

REFERENCES

[1] "Architecture of Windows Media Rights Manager", Microsoft Corporation, May 2004. http://www.microsoft.com/windows/windowsmedia/howto/articles/drmarchitecture.aspx

[2] R.Lee et al., "Architecture for protecting critical secrets in microprocessors," 32nd International Symposium on Computer Architecture (ISCA 2005), pp. 2-13, June 2005.

[3] Jeffrey S Dwoskin, Ruby B. Lee, "Hardware-rooted Trust for Secure Key Management and Transient Trust", *ACM Conference on Computer and Communications Security*, pp. 389-400, October 2007.

[4] Jeffrey Dwoskin, Dahai Xu, Jianwei Huang, Mung Chiang, Ruby Lee, "Secure Key Management Architecture Against Sensor-node Fabrication Attacks", *IEEE GlobeCom 2007*, November 2007.

[5] James Jeppesen et al., "Hard Disk Controller: the Disk Driver's Bain and Body", 0-7695-1200-3/01, 2001 IEEE.