A Security Analysis of Some Physical Content Distribution Systems

by

Jiayuan Sui

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2008

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Content distribution systems are essentially content protection systems that protect premium multimedia content from being illegally distributed. Physical content distribution systems form a subset of content distribution systems with which the content is distributed via physical media such as CDs, Blu-ray discs, etc.

This thesis studies physical content distribution systems. Specifically, we concentrate our study on the design and analysis of three key components of the system: broadcast encryption for stateless receivers, mutual authentication with key agreement, and traitor tracing. The context in which we study these components is the Advanced Access Content System (AACS). We identify weaknesses present in AACS, and we also propose improvements to make the original system more secure, flexible and efficient.

Acknowledgements

I would like to thank my supervisor, Dr. Douglas R. Stinson, for guiding me through the writing of this thesis. He has always been so patient and glad to help. I could not have completed this thesis without his insight and advice. I would also like to thank Dr. Ian Goldberg and Dr. Urs Hengartner for their careful reviews of the thesis as well as their constructive comments and ideas.

I am very grateful to my wife, Denae Clara Sui, for her love and unwavering support. I also owe many thanks to my parents for their constant help and encouragement over the years.

I am happy to acknowledge the David R. Cheriton School of Computer Science for its financial support during my study.

Last but certainly not least, I would like to thank all my friends for their support during my study.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Protecting multimedia content such as video and audio from piracy has been a challenging problem for the industry as well as for academia for quite some time. Multimedia content must be packaged and distributed to end users in such a way that no one other than a set of privileged users defined by the content owner should be able to use it. A system designed to achieve such a goal is known as a *content distribution system*. In addition, a content distribution system has to make sure that the privileged users do not have the ability to disseminate the content without permission while, at the same time, delivering pleasant user experiences.

Multimedia content is distributed from the content owner to the user via two channels: physical distribution and computer networking (online) distribution. To date, physical distribution is still a dominant method used by content owners, distributors, and users for trading premium multimedia content. In the physical distribution model, multimedia content is first packaged onto physical media (e.g. DVDs and CDs) for distribution. Upon receiving the media, the user can then play back the content using corresponding media players. Online distribution replaces physical media by means of telecommunications networks such as the Internet, and the content can be distributed much more cost-effectively. Multimedia data may be stored on a remote server and delivered to a client across a network using two techniques: The client first downloads the entire multimedia file containing audio or

video from a remote server to its local file system and then plays back the content, and a fairly modern technique known as streaming which enables the client to play back the content while the file is being downloaded. However, the advent of online distribution channels also enable quick and massive dissemination of illegal copies of multimedia content.

The two most common models exercised by the attacker to carry out piracy of multimedia content are: (1) the attacker compromises the distribution channel to obtain the multimedia content and then disseminates the multimedia content online, and (2) the attacker compromises the distribution channel to obtain some secret information which enables piracy of the multimedia content and then disseminates the secret information online. A number of cryptographic systems, such as *Content Scramble System* (CSS) for DVDs and *Extended Copy Protection* (XCP) for CDs, that have been developed for the purpose of retaining authorities' control over multimedia data, have not only failed to thwart piracy but have also raised usability issues. After more than a decade of research on the topic, designing a physical content distribution system that protects multimedia content from piracy, as well as provides positive user experience, remains a challenge.

In this thesis, we focus on studying three categories of cryptographic schemes that play significant roles in modern physical content distribution systems. They are broadcast encryption for stateless receivers, mutual authentication with key agreement, and traitor tracing. The context in which we will study these schemes is the *Advanced Access Content System* (AACS).

AACS is a physical content distribution system for recordable and pre-recorded media. It has been developed by eight companies: Disney, IBM, Intel, Matsushita (Panasonic), Microsoft, Sony, Toshiba, and Warner Brothers. Most notably, AACS is used to protect the next generation of high definition optical discs such as HD-DVD and Blu-ray discs. The current AACS technical specifications span nearly 550 pages spreading across seven documents [4, 5, 6, 7, 8, 9, 10], and they provide an overview of the system, as well as specific implementations for pre-recorded and

recordable media for both HD-DVD and Blu-ray formats. In this thesis, we only consider the part of AACS responsible for pre-recorded media.

The contributions of this thesis are:

(1) Provide a thorough explanation of the theory behind the three major components of a physical content distribution system, namely, broadcast encryption for stateless receivers, mutual authentication with key agreement, and traitor tracing in the context of AACS. This serves as a complement to the AACS specifications which provide little information on these cryptographic tools.

(2) Perform rigorous analyses of the three components to determine their efficiency and security. In the process, we identify weaknesses that are present in these components in AACS.

(3) Propose improvements on these components to make AACS as well as other physical content distribution systems using these schemes more efficient and secure.

The next chapters of this thesis are organized as follows:

- In Chapter 2, we present background knowledge. Specifically, we look into the structure of modern physical content distribution systems. We also briefly study two physical content distribution systems, CSS and *Content Protection for Pre-recordable Media* (CPPM), which are regarded as predecessors of AACS.

- In Chapter 3, we give an in-depth study of broadcast encryption schemes for stateless receivers. Stateless receivers are mainly media playback devices that do not have a constant online connection. They are the terminals of a physical distribution channel. A framework for designing broadcast encryption schemes for stateless receivers is presented. Much emphasis is placed on the study of AACS's broadcast encryption scheme, which is built on such a framework. We describe how the scheme works and how it is implemented in AACS.

We perform analyses on AACS's broadcast encryption to determine the efficiency of the scheme, and we also propose improvements aiming to enhance its efficiency. In the last part of the chapter, we demonstrate a technique to convert AACS's broadcast encryption scheme, which is symmetric-key, into a public-key scheme in order to achieve extra security features.

- In Chapter 4, we analyze AACS's drive-host authentication scheme, which is a mutual authentication protocol with key agreement. We observe a few flaws in the scheme, which could lead to various attacks on the system. We modify the original scheme into a new scheme, which provides secure mutual authentication as well as authenticated key agreement with key confirmation. A proof of security of our modified scheme is presented at the end of the chapter.

- In Chapter 5, we study how multimedia content can be modified to carry forensic data, which enables tracing to compromised devices so that they can be disconnected from the system. Our study concentrates specifically on combinatorial traitor tracing schemes, which use codes with certain structures to provide traceability. AACS is the first physical content distribution system to employ traitor tracing technology. We study how the AACS traitor tracing mechanism is designed and how it is implemented. We also provide analysis on the effectiveness of the AACS traitor tracing scheme.

- In Chapter 6, we give conclusions of our research along with a discussion of future work in making content distribution systems more robust against attackers who have full control of the execution environment.

Chapter 2

Background

In this chapter, we will present the basic structure of physical content distribution systems. We will introduce the roles broadcast encryption, authentication, and traitor tracing play in a physical content distribution system. Two earlier physical content distribution systems, Content Scramble System and Content Protection for Pre-recorded Media, will be studied briefly.

2.1 Basic Structure of Physical Content Distribution Systems

AACS and most other physical content distribution systems are elements of *Digital Rights Management* (DRM). DRM refers to technologies that support legal distribution of digital media and grant the end user appropriate power to access digital media. DRM technologies, however, do not dictate what content should be protected. According to Bosi [40], a DRM system needs to address three major areas:

(1) Ensuring that the multimedia content is packaged in a form that will prevent unauthorized copying/usage.

(2) Appropriately distributing the multimedia content and usage rules to the end user.

(3) Making sure the end user is able to access the multimedia content consistently with his/her rights. (End users have rights, e.g., fair use, which are not explicitly granted by the content provider.)

A physical content distribution system must carry the above three properties. Figure 2.1 shows the high-level structure of a physical content distribution system.

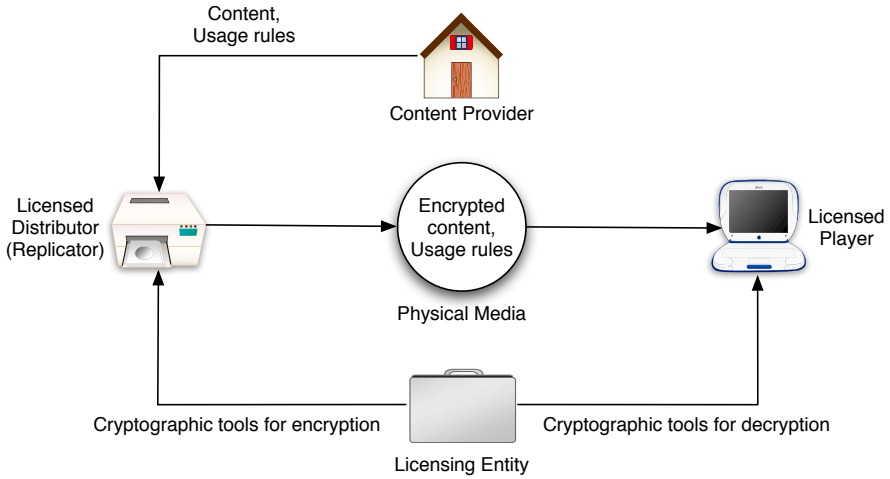

Figure 2.1: The basic structure of a physical content distribution system

The content provider (e.g., a movie studio) generates the multimedia content and the corresponding usage rules. Usage rules specify in what way the end user may access the content. The multimedia content and usage rules are sent to the licensed distributor for packaging and distribution; that is, encrypting the content and recording the encrypted content on physical media. The licensing entity is responsible for constructing cryptographic tools for the system. These tools form the core of the system that ensures content confidentiality and integrity. They are integrated into the licensed distributor and the licensed player to enable multimedia content encryption and decryption. The end user is also a part of the system. His/her licensed players should be able to play, according to the usage rules, any

packaged multimedia content ever broadcasted by the distributor to him/her. A more detailed structure regarding AACS is presented in Section 3.4.2.

Broadcast encryption is at the heart of almost all content distribution systems. Its responsibility is to facilitate the distribution of multimedia content in such a way that only a predefined set of users are able to play back the content. In the case of physical content distribution systems, licensed players are stateless, which means that they are difficult to update after the system is initialized, and the decryption of the broadcast is solely based on the player's initial configuration. The content encryption is performed by the licensed distributor, and the content decryption is performed by the end user's licensed player.

Some broadcast encryption schemes have the ability to revoke certain licensed players in the event that these players are compromised by an attacker. Once these players are revoked, they are no longer capable of decrypting any newly released broadcast. Therefore, the attacker can no longer use the secret information stored in the players to pirate any newly released multimedia content. Traitor tracing mechanisms are developed to enable the authority to find out which players are compromised based on forensic data (e.g., pirate decoders or pirated content), and then broadcast encryption can revoke those players.

Authentication is required at a few places in a physical content distribution system, two of which are most notable. The first one is that the pre-recorded media must be able to authenticate itself to the licensed player before content playback. This authentication procedure is required for every pre-recorded medium in order to assure that the content on the medium has not been illegally altered and the content itself has not been revoked by the licensing entity. The other authentication is required in the case that the licensed player is a PC. Before the content can be played, the drive (e.g., DVD-ROM) and the host (e.g., player software) of a PC must be able to mutually authenticate to each other. This mutual authentication procedure ensures that both the drive and the host are in compliance with the licensing entity's specifications.

In the next sections, we briefly study two of the most influential physical content distribution systems for pre-recorded media.

2.2 Content Scramble System

Content Scramble System (CSS) is a physical content distribution system used on Digital Versatile Discs (DVDs) [51]. It was initiated by the DVD Copy Control Association (CCA) and released in 1996. CSS follows the basic structure of a physical content distribution system. The movie studio provides the content and usage rules to the licensed replicator for distribution. The DVD CCA is the licensing entity that provides cryptographic tools to the distributor for content encryption. It also provides cryptographic tools to the device manufacturer to be included into DVD players for content decryption.

Figure 2.2 shows an overview of how CSS works for DVDs. Multimedia content is provided by the content provider in forms of *titles*. For simplicity, we can think of a title as a movie. A title is partitioned into sectors of data. Each sector is of size 2048 bytes, the first 128 bytes of which is the sector's header. The header does not include any of the multimedia content. A 40-bit key specific to the title called *title key* is exclusive-ORed with bytes 80-84 of the header to produce a *sector key*. The sector key is then used to encrypt the data in the sector excluding the header. All the sectors of the title are encrypted using the same method and burnt onto a DVD.

The title key is in turn encrypted with a disc-specific *disc key* (K_d), which has length 40 bits, and the disc key is then encrypted with all the *player keys*. The resulting encrypted disc keys are stored in a table. In CSS, there are a total of 409 player keys, each of which again has length 40 bits. Every DVD manufacturer is given a unique subset of them to be included in their DVD players. The purpose of doing so is that, since every DVD manufacturer has a different subset of player keys, if a manufacturer leaks its subset of player keys, it can be tracked down. A disc key

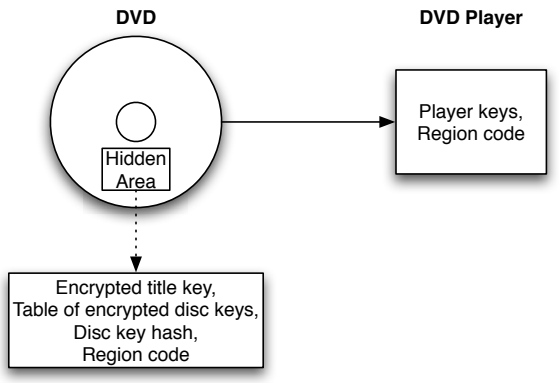

Figure 2.2: Overview of CSS

hash (H) is produced by encrypting the disc key with itself: $H = E_{K_d}(K_d)$ where E is an encryption algorithm used by CSS. The hash is later used by the player to verify the decryption of the encrypted disc key. A region code is also designated to the DVD which prescribes in which region (e.g., North America, Europe, Asia, etc.) the DVD is supposed to be used. In the end, the encrypted title key, the table of encrypted disc keys, the disc key hash, and the region code are all burnt onto the DVD's "hidden area". The contents of this hidden area cannot be delivered, except to an authenticated device. The hidden area of the DVD stores all the necessary information that enables the decryption of the encrypted content on the DVD. Consumer-writable DVDs have this area marked as unwritable.

Once the DVD is distributed to a user, the user can use his/her licensed player to play it. The process of playing the DVD requires the following steps:

(1) The region code on the DVD is compared with the region code on the DVD player. If they do not match, the player should stop processing the DVD any further. The region code is designed to implement *regional lockout* which, in the case of DVDs, facilitates the MPAA (Motion Picture Association of America) to control various aspects of the release (e.g., release date, price, etc) of the DVD in different regions in the world.

(2) The DVD player decrypts one of the 409 encrypted disc keys with one of its

player keys. The decrypted disc key K'_d is verified with the hash stored in the hidden area on the DVD: $K'_d \stackrel{?}{=} D_{K'_d}(H)$, where D is the decryption algorithm corresponding to E.

(3) The DVD player uses the disc key to decrypt the encrypted title key.

(4) For a sector received by the DVD player from the DVD, the DVD player calculates the sector key for the sector using the title key and bytes 80-84 of the sector's header.

(5) The DVD player uses the sector key to decrypt the encrypted multimedia content stored in the sector. The DVD player is then able to play back the content.

It is an unarguable fact that CSS has failed to protect DVDs from piracy. The most well-known tool that can effectively circumvent CSS, DeCSS (CSS Descrambler), has been available since 1999 [109], although a few methods to pirate DVDs had existed before it. A number of weaknesses of CSS have led to various successful attacks to the system. Here are just a few:

- CSS's 40-bit keys are too short. Exhaustive search attacks are possible. The attacker can simply try each of all the possible 2^{40} disc keys on $E_{K_d}(K_d)$ and check if the result is equal to the value of H stored on the disc. If so, there is a high probability that the correct disc key is found. Faster attacks have been studied, bringing the time complexity to 2^{25} and even lower [100]. The short key length was chosen to satisfy U.S. government export regulations [75]. (This restriction had been lifted when AACS specifications were created.)

- CSS does not have flexible key management (i.e., broadcast encryption is weak). If a set of player keys are extracted from a DVD player by the attacker, any DVD can be decrypted using these compromised keys. The DVD CCA can do little about revoking the leaked keys, because revoking the keys would render many innocent DVD players useless. Moreover, determining which keys have been compromised would also be challenging given that CSS does not have a traitor-tracing capability.

- CSS does not effectively prevent bit-by-bit copying of DVDs. Attackers with DVD pressing equipments can replicate DVDs in large volume within a short time and then distribute them illegally.

Besides the encryption and decryption procedures we have discussed above, CSS also has a weak drive-host authentication protocol with key agreement to protect data transfered between the drive and the host. For more information on CSS, please refer to the specification [51].

2.3 Content Protection for Pre-recorded Media

Content Protection for Pre-recorded Media (CPPM) is a physical content distribution system primarily used for protecting DVD-Audio. It has a counterpart for recordable media, *Content Protection for Recordable Media* (CPRM), which is widely used in Secure Digital cards. CPPM/CPRM was released in 2000 by the 4C Entity, LLC, which is a consortium founded by IBM, Intel, Matsushita, and Toshiba.

CPPM is an upgrade from CSS. It features a much better broadcast encryption scheme which empowers the system with renewability by revoking compromised devices. The revoked player will not have the ability to play back media after the revocation. In addition, the system delivers the following features according to its specification [1]:

- It meets the content owners' requirements for robustness.
- It is applicable to both audio and video content.
- It is equally suitable for implementation on PCs and Consumer Electronics (CE) devices.
- It is applicable to different read-only media types.

In the structure of CPPM, the 4C Entity, LLC is the licensing entity that provides cryptographic tools to the licensed replicator and the device manufacturer. As shown in Figure 2.3, the system works as follows:

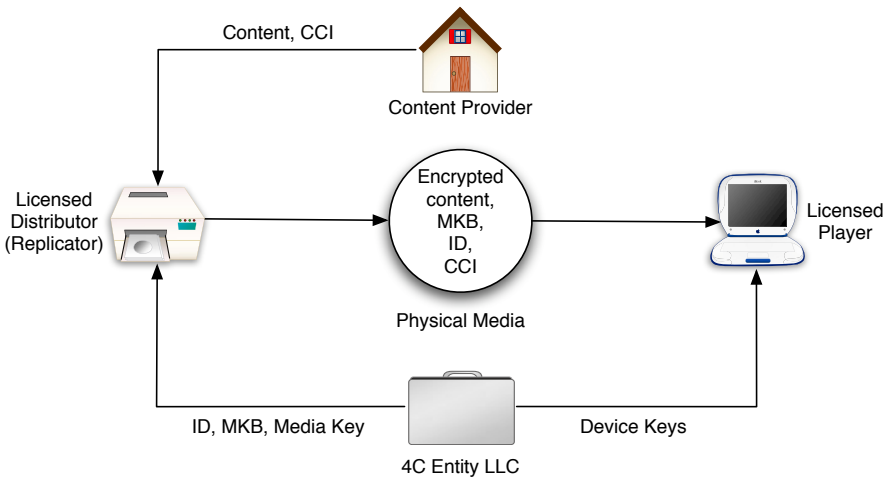

Figure 2.3: Overview of CPPM

(1) The content owner provides the multimedia content (a title) to a licensed replicator for distribution. For this content, the replicator receives a unique title-specific *identifier* (ID), a *media key block* (MKB), and a *media key* K_m corresponding to the MKB from the 4C Entity, LLC.

(2) The replicator computes the *content key* $K_c = \text{Hash}(K_m, \text{ID}, \text{CCI})$ from the media key, the ID, and the CCI (*Copy Control Information*, which determines the copy control status of the content) using a one-way hash function. The content key is used to encrypt the content. The replicator then records the encrypted content, the MKB, the ID, and the CCI on the medium. The ID is embossed in a read-only section of the disc (e.g., the burst cutting area on an optical disc) to prevent bit-by-bit copying of the media content onto rewritable media via consumer disc burners.

(3) The device manufacturer receives a number of secret *device keys* to be included into each player produced. The device manufacturer assigns a set of n device

keys to each device. A key set may be either unique per device, or shared by a group of devices. The n device keys are used to process the MKB on the medium in order to derive the media key K_m; we will see the details of this in the next section.

(4) When a CPPM protected medium is inserted into a compliant playback device, the device uses its set of device keys to process the MKB on the medium to obtain the media key K_m. The device then reads the ID and the CCI off the medium and computes the content key $K_c = \text{Hash}(K_m, \text{ID}, \text{CCI})$. Finally, the encrypted content is decrypted using the content key and then played back by the device.

Arguably, the most significant improvement of CPPM over CSS is its broadcast encryption, which allows compromised devices to be revoked without affecting other innocent devices' ability to play back the media. In the following section, we will take a closer look at CPPM's broadcast encryption.

2.3.1 CPPM Broadcast Encryption

The broadcast encryption scheme employed by CPPM is designed for systems with stateless receivers. Each receiver device or each model of devices is assigned with a unique set of n device keys. For simplicity, we assume that each set of device keys is unique per device. The device keys are assigned from an $m \times n$ table with each entry containing a unique device key. A set of n device keys for a device are assigned by choosing one of the m device keys from each column, and this set of device keys must be able to process the MKB to derive the corresponding K_m. No two sets of device keys are the same, although it is possible that some sets have a certain amount of device keys in common. If a set of device keys is compromised, it can be revoked so that using it to process the MKB would not yield K_m.

Unfortunately, how CPPM chooses a device key from a column is not defined in the available specifications. A number of key assignment strategies are proposed

by Adelsbach and Schwenk [96]. In this section, we use a trivial key assignment strategy to illustrate how the system works. Basically, we assign each possible key combination across the columns to a device as a legitimate set of device keys. The example we use is similar to the one in [96]. Suppose that we have a 3×2 device key table with six device keys:

$K_{1,1}$	$K_{2,1}$
$K_{1,2}$	$K_{2,2}$
$K_{1,3}$	$K_{2,3}$

Each device key in the first column can be combined with any key in the second column to form a set of device keys for a device. Hence, this scheme can accommodate a total of nine devices. Pictorially, figure 2.4 shows the key assignment of the nine devices. Each black point represents a device, and each line represents a

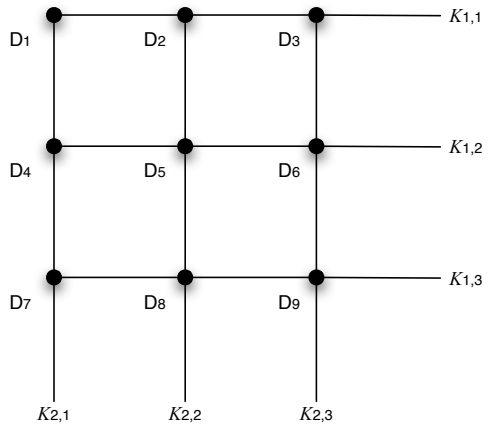

Figure 2.4: A trivial key assignment strategy for CPPM [96]

device key. A point on a line indicates that the device is assigned with the device key represented by that line. For example, device D_3 is assigned with $K_{1,1}$ and $K_{2,3}$. If the key table had three columns, the representation of key assignment would be 3-dimensional.

If there is no compromised device in the system, the media key K_m will be encrypted with every device key in a column of the device key table. Without loss

of generality, we use device keys in the first column. The encrypted media keys are stored in the *Calculate Media Key Record* (CMKR), which is put into the MKB:

Calculate Media Key Record
Column 1
$K_{1,1}(K_m)$
$K_{1,2}(K_m)$
$K_{1,3}(K_m)$

When a device processes the MKB, it has no problem decrypting the media key because any device in the system has one of $K_{1,1}$, $K_{1,2}$, and $K_{1,3}$.

The scheme allows some compromised devices to be revoked without affecting other devices' ability of processing the MKB. For example, devices D_1 and D_3 are compromised, and therefore $K_{1,1}$, $K_{2,1}$, and $K_{2,3}$ are known to the attacker. The white points in figure 2.5 represents the compromised devices. In this case, CPPM

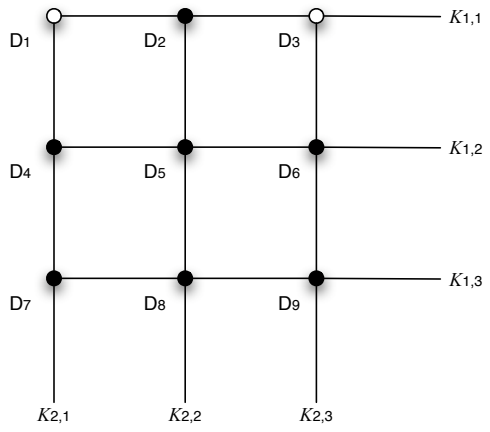

Figure 2.5: Revocation of D_1 and D_3

introduces *Conditionally Calculate Media Key Record* (CCMKR) to implement the revocation mechanism. If any device key in the first column of the device key table is compromised, instead of using it to encrypt the media key, it is used to encrypt a *link key* K_l. Otherwise, the device key is used to encrypt the media key. The ciphertexts are then included in the CMKR. Just by processing the CMKR,

devices D_1, D_2, and D_3 cannot obtain the media key. In order not to discriminate against D_2 which is an innocent device, the system makes use of the device keys in the second column of the device key table. If any device key in the second column is compromised, it encrypts a "null" value. Otherwise, it encrypts the media key. Each encrypted media key/null is again encrypted with the link key from the previous column. The double-encrypted media key/null values are stored in a CCMKR. The column number of the CCMKR is also encrypted with the link key. Hence, the ability of processing the CCMKR is based on the condition of having the correct link key.

Calculate Media Key Record	Conditionally Calculate Media Key Record
Column 1	K_l(Column 2)
$K_{1,1}(K_l)$	$K_l(K_{2,1}(\text{null}))$
$K_{1,2}(K_m)$	$K_l(K_{2,2}(K_m))$
$K_{1,3}(K_m)$	$K_l(K_{2,3}(\text{null}))$

CMKR and CCMKR are both included in the MKB. Device D_2, which has device key $K_{1,1}$ and $K_{2,2}$ will obtain the media key by processing the MKB. However, the compromised devices D_1 and D_3 will only get the "null" value. Depending on the number of columns in the device key table and the status of the system, there might be many CCMKRs in the MKB, whereas there is always just one CMKR.

This trivial key assignment strategy maximizes the number of devices that the system can accommodate. For an $m \times n$ device key table, the system can accommodate m^n devices. However, this scheme is very weak against coalition attacks. For instance, as Figure 2.6 suggests, if devices D_1, D_5, and D_9 are compromised, they collectively know all the device keys in the system. The system is thus broken. Moreover, some devices can collude together to "frame" certain innocent devices such that revocation of the compromised devices can render some innocent devices useless. As shown in Figure 2.7, D_1 and D_9 collectively know $K_{1,1}$, $K_{2,1}$, $K_{1,3}$, and $K_{2,3}$. If these two devices are compromised and the corresponding device keys are revoked, D_3 and D_7 would suffer collateral damage and would not be able to obtain

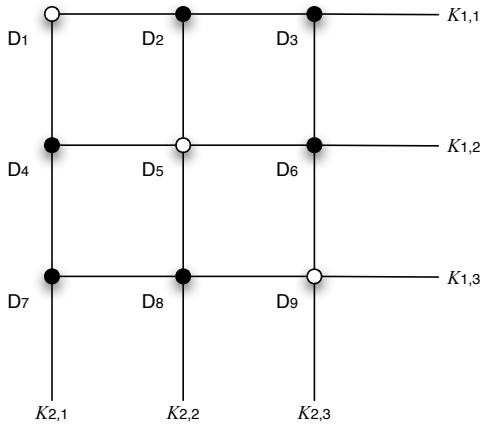

Figure 2.6: D_1, D_5, and D_9 together can compromise the whole system

the media key.

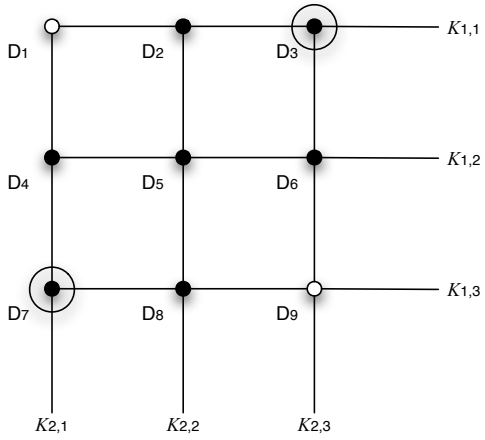

Figure 2.7: Framing of D_3 and D_7 by D_1 and D_9

There is another trivial key assignment strategy for CPPM broadcast encryption that is quite the opposite of the one we have just studied. Each device is assigned with a set of device keys in a row of the device key table. For an $m \times n$ device key table (in this case, we can set $n = 1$), the system can only accommodate m devices. The advantage is that the system achieves *full-collusion resistance*, which means that no matter how many revoked devices collude together they cannot decrypt the broadcast, and the innocent devices' ability to obtain the media key is unaffected. Neither of the two trivial strategies are practical for large content

distribution systems such as CPPM.

CPPM does not have a traitor-tracing capability, and it is unclear how the licensing authority learns which devices are compromised in the first place. The structure of CPPM broadcast encryption is employed by AACS traitor tracing scheme to achieve renewable tracing, which we will study in Chapter 5. CPPM employs a similar drive-host authentication protocol as CSS. For more information on CPPM, please refer to the specifications [1, 2].

Chapter 3

Broadcast Encryption for Stateless Receivers

3.1 Introduction

Broadcast encryption is gaining increasing popularity in commercial applications such as AACS, Pay-TV, etc. It provides a convenient way to distribute digital content to subscribers over an insecure broadcast channel. This cryptographic tool is used by a broadcast center (or trusted authority) to distribute an encrypted message to a set of users in such a way that allows only qualified users (e.g., subscribers who have paid the fee and behaved according to the agreements) to decrypt the encrypted message, excluding all others.

Let us use $\mathcal{U}$ to denote the set of n users in a broadcast encryption scheme,

$$\mathcal{U} = \{u_1, u_2, \ldots, u_n\}.$$

The encrypted message is broadcast to $\mathcal{U}$ by the broadcast center. Only a subset of $\mathcal{U}$, denoted $\mathcal{P}$, can decrypt the encrypted message. We refer to $\mathcal{P}$ as the set of privileged users. The users in $\mathcal{U}$ that can not decrypt the encrypted message comprise the set of revoked users, denoted by $\mathcal{R}$. Clearly, $\mathcal{P} = \mathcal{U} \setminus \mathcal{R}$, $\mathcal{P} \cup \mathcal{R} = \mathcal{U}$, and $\mathcal{P} \cap \mathcal{R} = \emptyset$.

Most broadcast encryption schemes, including the scheme used in AACS, can be divided into three steps:

(1) An initialization step assigns each user in $\mathcal{U}$ some secret information (e.g., a set of keys) that allows the user to decrypt the encrypted message.

(2) The broadcast center uses a broadcast encryption algorithm, which takes as an input a message $\mathcal{M}$ and a set of users $\mathcal{R}$ whose decryption privileges are revoked, and outputs an encrypted message. The encrypted message is then broadcasted to $\mathcal{U}$.

(3) Upon receiving the broadcast, a user can use his/her secret information to decrypt the encrypted message to obtain $\mathcal{M}$ if and only if he/she is in $\mathcal{P}$.

Since this broadcast encryption model is a function that takes as an input a set of revoked users $\mathcal{R}$, it is also referred to as *Revocation Scheme*.

The designer of a broadcast encryption scheme has to take into consideration the various conditions and constraints present in the broadcast center, the receiver, and the communication channel. Both the center and the receiver have storage and computational limitations. The communication channel has bandwidth limitations. Different types of receivers support different functionalities. Some types of receivers can be easily reached from the center and their firmware can be updated by the center quickly without a problem, whereas some others may find that it is unnecessary or undesirable to have such features.

The revocation scheme used by AACS assumes "stateless receivers". In such a scheme, receivers are "off-line". Once the secret information is assigned to the receiver, it becomes very hard to update throughout the lifetime of the system. The size of the user population (the number of receivers) of such a scheme is usually pre-determined, since it is very inconvenient to change the population size after the system is initialized. Communications between the center and stateless receivers are usually unidirectional in which case the information flows only from the center

to the receivers and it is hard for the receivers to send messages to the center. Hence, each broadcast must be decrypted solely on the basis of the fixed initial configuration of the receiver. On the other hand, "stateful receivers" are much more flexible, as the center can update their secret information regularly with ease; however, this might require a continuous on-line connection. Applications employing stateful receivers are, for example, Pay-TV, audio streaming, real-time business data, multicast communication, etc. The merit of the "stateless receivers" model is that it allows the hardware and the software at the client end to be as simple as possible. It is particularly popular with regard to storage media for multimedia content.

In this chapter, we focus on the study of broadcast encryption regarding DRM for stateless receivers of unidirectional communication, especially the revocation scheme used by AACS. We give theoretical as well as implementation-related analyses of AACS's revocation scheme, and identify weaknesses present in the scheme. Possible modifications of the scheme are suggested at the end, aiming to make the scheme more efficient.

3.2 Related Work

The phrase "broadcast encryption" was first coined by Fiat and Naor in their 1993 paper [57]. Since then it has become an active research area. The results from this paper are called *k-resilient schemes*. The factor k is the maximum number of colluding revoked users against which the scheme is secure. If k is equal to the number of revoked users then we say that the scheme is *full-collusion resistant*, which means that no matter how many revoked users collude together they cannot decrypt the encrypted broadcast message. The paper first gives two constructions of 1-resilient schemes based on one-way functions and extracting roots modulo composites, respectively. k-resilient schemes can then be constructed by combining 1-resilient schemes using perfect hash functions. Their best scheme has

storage complexity of $O(k \log k \log n)$ for the receiver and the center broadcasts $O(k^2 \log^2 k \log n)$ keys. In practice, a reasonably sized k is usually big enough to make such schemes impractical to use. Further work includes [15], [52], [102], [103].

The logical-tree-hierarchy (LKH) scheme, suggested independently by Wallner et al. [116] and Wong et al. [118], is designed for stateful receivers. The scheme is full-collusion resistant, and the receivers are active participants who can request, join or leave at any time. LKH is a tree-based revocation scheme where a user is represented by a leaf of an a-ary tree. The center allocates keys to each node in the tree, and the user receives all the keys on the path from itself to the root. When a user is revoked, all remaining users' secret keys have to be updated. It requires a transmission of $2r \log n$ keys to revoke r users, and the storage complexity for the user is $O(\log n)$. The message is encrypted by a session key which in turn is encrypted by the secret key corresponding to the root. The encrypted message and the encrypted session key are broadcasted to the users. This scheme has low bandwidth complexity because besides the message only the encrypted session key needs to be broadcasted. Improvements based on this scheme are discussed in [41], [43], [84].

Noar et al. [89] introduced two revocation schemes for stateless receivers. They are also tree-based schemes and full-collusion resistant. The first, "complete subtree revocation scheme", is very similar to LKH. Since it is for stateless receivers, the number of users is determined before the initialization of the scheme. The users are the leaves of a binary tree, and all users are divided into subsets based on the subtrees in the entire binary tree. Every node is assigned a key by the center, and all the users within a particular subtree get the key corresponding to the root of the subtree. A subset-cover is formed to keep track of the subsets consisting of all the privileged users. The storage complexity at the receiver's end is $O(\log n)$ and the complexity of the message expansion is $O(r \log \frac{n}{r})$. The second scheme in the paper is "subset difference revocation scheme" (SD) which is used by AACS as its broadcast encryption scheme. This one has better complexity of the message

expansion than the complete subtree revocation scheme. We will study this scheme in detail in Section 3.3. Halevy and Shamir [63] generalized the subset difference revocation scheme to "layered subset difference scheme" (LSD). This scheme is described in detail in section 3.5.1.

The schemes mentioned above are all symmetric-key schemes. A few broadcast encryption schemes based on public-key techniques have also been developed. A few of the relevant works are: [36], [54], [55], [79], [90] and [117]. In public-key broadcast encryption schemes, messages are encrypted with public keys, so everyone can broadcast information. This feature cannot be achieved in the symmetric-key setting due to the fact that only the broadcast center knows the secret keys which are necessary for broadcasting messages. A broadcast center in a symmetric-key broadcast encryption scheme has to store all the secret keys of all the users of the system, which makes it a single point of failure. On the other hand, a broadcast center in a public-key setting only needs to know the public key information. We will show in Section 3.5.2 how to convert the subset difference revocation scheme from its original symmetric-key setting into a public-key scheme.

3.3 Subset Difference Revocation Scheme

3.3.1 Subset-Cover Framework

Naor et al. have suggested a subset-cover framework [89], which serves as a nice prototype for most revocation schemes, including the one employed by AACS. In this framework, a set of non-empty subsets of $\mathcal{U}$, $S = \{S_1, S_2, \ldots, S_w\}$, $S_j \subseteq \mathcal{U}$, is defined such that $\mathcal{U} = \bigcup_{j=1}^{w} S_j$. Subsets need not be disjoint; a single user may be in many subsets. A *long-lived key* L_j is assigned to each subset S_j. A user u in S_j should be able to deduce L_j easily from his/her secret information. If S is constructed in an appropriate way, then for any non-empty set of privileged users

$\mathcal{P}$ we can find a set of subsets $C = \{S_{i_1}, S_{i_2}, \ldots, S_{i_m}\}$ that exactly "covers" $\mathcal{P}$, i.e.

$$\mathcal{P} = \mathcal{U} \setminus \mathcal{R} = \bigcup_{j=1}^{m} S_{i_j}.$$

Note that $C \subseteq S$, and is called a subset-cover for $\mathcal{P}$. We can then encrypt a *session key*, K, m times with $L_{i_1}, L_{i_2}, \ldots, L_{i_m}$, where K is used to encrypt the message M.

A session key is used only for a "short term", therefore the common practice is that a different session key is used for each different broadcast. A long-lived key, on the other hand, is designed to be used throughout the lifetime of the system. The main reason why session keys are useful in broadcast encryption is to reduce data complexity. If the message is encrypted directly with the long-lived keys, to remove the need for a session key, then the length of the broadcast would be the length of the message times the size of the cover. If the size of the message is large, which is usually the case for multi-media content distribution, then it is almost impossible to make a broadcast in this way.

As we have already mentioned in the previous section, there are three steps in a revocation scheme, namely, initialization, broadcast encryption, and broadcast decryption. The subset-cover framework follows these three steps:

Initialization: A user $u \in \mathcal{U}$ is given a piece of secret information I_u by the broadcast center. I_u is used to deduce the long-lived key L_j corresponding to the subset S_j in which he/she is included. If a user is in many subsets, then I_u should be able to allow him/her to deduce all the long-lived keys for those subsets. If each of the keys L_j are generated uniformly at random and independently from each other, we say that the keys are chosen *information-theoretically*. On the other hand, if the keys are obtained as a function of some secret information, then the keys are generated *computationally*. Using computational key generation instead of information-theoretic key generation can greatly reduce the space requirements for key storage; however, it sacrifices information-theoretic security for computational security. In this framework, I_u can be a set of long-lived keys $\{L_{u_1}, L_{u_2}, \ldots, L_{u_k}\}$ (information-theoretic),

or it can be an input to a function which outputs a set of keys (computational).

Broadcast encryption: Given a message M and a set of revoked users $\mathcal{R}$, the broadcast center first chooses a session key K with which to encrypt M. The center then finds a subset-cover $C = \{S_{i_1}, S_{i_2}, \ldots, S_{i_m}\}$ which covers the set of non-revoked users $\mathcal{P}$. The long-lived keys $\{L_{i_1}, L_{i_2}, \ldots, L_{i_m}\}$ corresponding to the subsets in the subset-cover are then used to encrypt the session key. The final broadcast made by the broadcast center to $\mathcal{U}$ is

$$\langle \underbrace{i_1, i_2, \ldots, i_m, E_{L_{i_1}}(K), E_{L_{i_2}}(K), \ldots, E_{L_{i_m}}(K)}_{header}; E_K(M) \rangle.$$

The portion that precedes the semicolon is called the header of the broadcast, and $E_K(M)$ is termed the body of the broadcast. The size of the header, measured in the number of indices and encrypted session keys, is referred to as the broadcast message expansion.

Broadcast decryption: A user u is given all the indices of the subset differences in which he/she is a member. Upon receiving the broadcast, u can first search through the list of indices, $\{i_1, i_2, \ldots, i_m\}$, to find an index i_j that he/she has. If such an index can not be found, then u is revoked. The user u then extracts the corresponding key L_{i_j} from I_u and uses L_{i_j} to decrypt the corresponding encrypted session key, $D_{L_{i_j}}(E_{L_{i_j}}(K)) = K$. Finally, u uses K to obtain the message, $D_K(E_K(M)) = M$.

To illustrate how revocation schemes can be built based on this framework, we present two trivial revocation schemes, A and B, along with some comments regarding their impracticalities.

Trivial A: Let $\mathcal{U} = \{u_1, u_2, \ldots, u_n\}$ be a set of users. The broadcast center constructs S in a way such that each subset contains only one user, that is, $S = \{\{u_1\}, \{u_2\}, \ldots, \{u_n\}\}$. A long-lived key is assigned to each subset, which is to say that each user u_i is given one long-lived key L_i. Whenever

the broadcast center wants to send a message to a set of privileged users $\mathcal{P}$, it encrypts the message using a session key and then encrypts the session key with the key L_i, for all i such that $u_i \in \mathcal{P}$. Upon the reception of the broadcast, a privileged user uses his/her long-lived key to compute the session key and then uses the session key to obtain the message.

A nice property of this scheme is that each user is required to store only a single key, making the storage size is optimal. However, the message size grows linearly with respect to the size of $\mathcal{P}$. If the broadcast center needs to send a message to one million recipients, the message expansion would be of two million entries in the header, which in most cases is an unrealistic requirement.

Trivial B: Again, let $\mathcal{U} = \{u_1, u_2, \ldots, u_n\}$ be the set of users. The set S is constructed in such a way that it includes every possible non-empty subset of $\mathcal{U}$, i.e. $S = 2^{\mathcal{U}} \setminus \emptyset$. A long-lived key is assigned to each subset, hence there are $2^n - 1$ long-lived keys in total. When the broadcast center sends a message to a set of privileged users $\mathcal{P}$, it finds the subset S_j that is equal to $\mathcal{P}$ and encrypts the session key using the long-lived key L_j corresponding to S_j. The final broadcast is simply $\langle j, E_{L_j}(K); E_K(M)\rangle$. The decryption process for the users in $\mathcal{P}$ should be straightforward.

This scheme is essentially the opposite of the previous scheme. The message expansion is always 2, which is optimal; however each user must store a large number of keys. In the case of $|\mathcal{U}| = n$, each user must store $2^n - 1$ keys. Thus, with only a few dozen users this scheme becomes completely impractical.

There is a trade-off between broadcast message expansion and the number of keys the user is required to store. The challenge is to find a revocation scheme that gives a proper balance between the two. A fairly efficient revocation scheme (used by AACS) is discussed in the next section.

Any revocation scheme that is developed based on this subset-cover framework

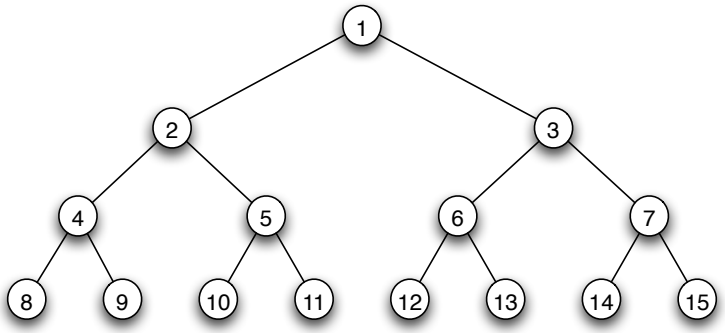

Figure 3.1: A complete binary tree with eight leaves

achieves semantic security against non-adaptive chosen-ciphertext attack (IND-CCA-1). For the details of the security analysis regarding the subset-cover framework, please refer to Naor et al. [89].

3.3.2 Description of Subset Difference Revocation Scheme

Naor et al. have proposed the following subset difference revocation scheme along with the subset-cover framework which was introduced in the previous section [89]. A PhD thesis by Martin also explains the SD scheme in detail [83].

The SD scheme is a tree-based revocation scheme. Let $\mathcal{U} = \{u_1, u_2, \ldots, u_n\}$ be a set of n users where $n = 2^h$, such that each user corresponds to a leaf node in a complete binary tree of height h. If $n \neq 2^h$, then we can use the smallest complete binary tree with more than n leaves. The complete binary tree has $2n - 1$ nodes, labeled breadth-first as $i = 1 \ldots 2n - 1$. Figure 3.1 shows a complete binary tree with eight leaves where nodes 8 through 15 correspond to the eight users.

We form S to be a set of differences between two subsets of $\mathcal{U}$. A difference between two subsets of $\mathcal{U}$, which is also a subset, is denoted $\gamma(i, j) = \mathit{desc}(i) \setminus \mathit{desc}(j)$. $\mathit{desc}(i)$ represents the set of leaf nodes in the subtree rooted at i. Note that j is a descendant of i. i is called the *primary node* and j is called the *secondary node* of this subset difference. There is a special case: $\gamma(0, 0) = \mathcal{U}$. Clearly, every

$\gamma(i,j)$ is a subset of $\mathcal{U}$. Figure 3.2 gives an example of $\gamma(2,5)$. Hence,

$$S = \left\{ \gamma(i,j) \mid \left(1 \leq i,j \leq 2n-1 \text{ and } desc\,(j) \subsetneq desc\,(i) \right) \text{ or } (i,j) = (0,0) \right\}.$$

In the case of information-theoretic key assignment, each element $\gamma(i,j)$ in S is assigned a long-lived key $L_{i,j}$, and each user in $\gamma(i,j)$ has a copy of $L_{i,j}$. Letting $\mathcal{R}$ be a set of revoked users, a fairly straightforward algorithm (Algorithm 1) is given to find the subset-cover for $\mathcal{U} \setminus \mathcal{R}$.

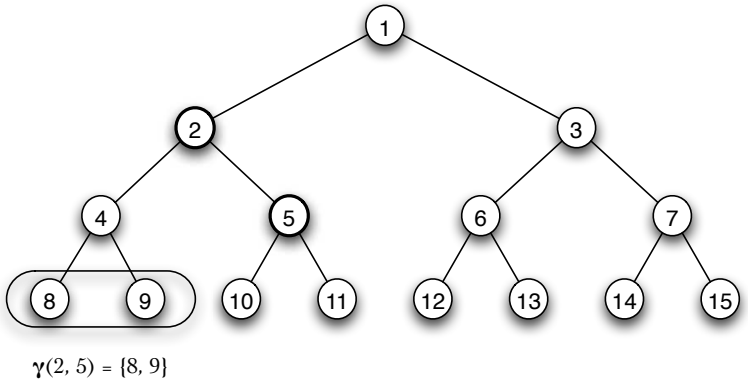

Figure 3.2: A set of leaf nodes represented by $\gamma(2,5)$

In line 1.3 in Algorithm 1, we need to find the least common ancestor for two given leaves. This is the *Least Common Ancestor* (LCA) problem, which can be solved in $O(m \log m)$ [24], where m is the number of nodes in the Steiner tree $ST(R)$ (a Steiner tree is a minimum spanning tree connecting a designated set of nodes [65]). After finding the least common ancestor of two given leaves, we can use depth-first search to test if this ancestor has no other leaves in $ST(R)$. Depth-first search has complexity $O(b^d)$ where b and d are the branch factor (at most 2) and the depth of the subtree rooted at this ancestor. To efficiently find one pair of leaves out of the $\binom{r}{2}$ possible pairs to satisfy line 1.3, we can use the heuristic that the least common ancestor of two leaves that are close to each other in $ST(R)$ has a better chance to have no other leaves in $ST(R)$. In the worst case, the algorithm needs to test all the $\binom{r}{2}$ possible pairs in order to find a right pair that satisfy line 1.3, which results in $O(r^2)$ complexity. Each iteration through the

Algorithm 1: Subset-Cover Algorithm

1.1 Initialize the subset-cover C to an empty set, and let $ST(R)$ be the Steiner tree induced by $\mathcal{R}$ and the root 1.

1.2 **while** $ST(R)$ *has more than one leaf* **do**

1.3 Find two leaves i and j such that their least common ancestor, v, has no other leaf descendants in $ST(R)$;

1.4 Let k and l be children of v such that i is a descendant of k and j is a descendant of l;

1.5 **if** $k \neq i$ **then**

1.6 $C = C \cup \{\gamma(k, i)\}$;

1.7 **end**

1.8 **if** $l \neq j$ **then**

1.9 $C = C \cup \{\gamma(l, j)\}$;

1.10 **end**

1.11 Remove everything descended from v in $ST(R)$ and make v a leaf node of the new $ST(R)$;

1.12 **end**

1.13 **if** *the only leaf* i *of* $ST(R)$ *is not the root* 1 **then**

1.14 $C = C \cup \{\gamma(1, i)\}$;

1.15 **end**

while loop reduces the number of Steiner leaves by one, hence there are a total of $r - 1$ iterations for the while loop. Other operations in the while loop are not as expensive as the one on line 1.3. The complexity of Algorithm 1 in the worst case is $O(r^3 * (m \log m + 2^d))$.

After executing this algorithm, $C = \{\gamma(i_1, j_1), \ldots, \gamma(i_m, j_m)\}$ is the subset-cover for $\mathcal{U} \setminus \mathcal{R}$. Upon finding C, the broadcast center can simply create the broadcast

$$\langle (i_1, j_1), \ldots, (i_m, j_m), E_{L_{i_1,j_1}}(K), \ldots, E_{L_{i_m,j_m}}(K); E_K(M) \rangle$$

and send it to $\mathcal{U}$. Figure 3.3 shows an example in which three leaf nodes are revoked, and $ST(R)$ is highlighted. Using the above algorithm, we find that the subset-cover for the remaining leaf nodes is $\{\gamma(2,8), \gamma(3,6)\}$.

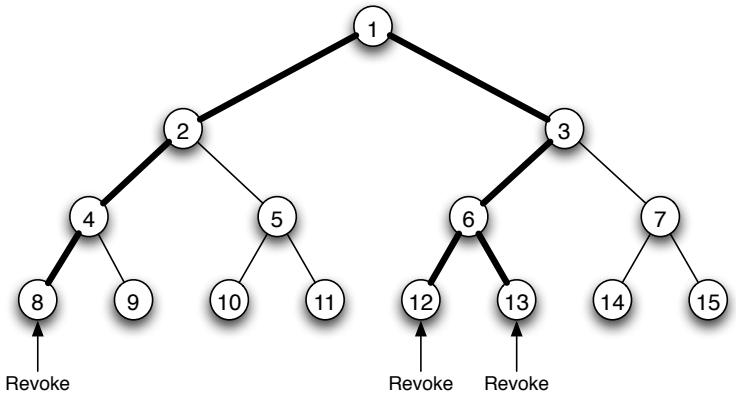

Figure 3.3: A Steiner subtree induced by the root and the three revoked leaves

3.3.3 Analysis of Subset Difference Revocation Scheme

Complexity of the message expansion and storage complexities at both the server and the client ends are essential factors in determining the efficiency of a broadcast encryption scheme. We will study these factors for the SD scheme in two settings: One with keys assigned uniformly at random and the other one with keys obtained as a function of some secret information.

Information-theoretic key assignment

(1) **Complexity of the message expansion:**

In Algorithm 1, each iteration through the while loop increases the number of subsets in C by at most 2, while reducing the number of Steiner leaves by 1. At the final step (step 3), when there is only one leaf, at most one subset is added to C. Hence, given any set of revoked users, $\mathcal{R}$ with cardinality r, the size of the subset-cover for $\mathcal{U} \setminus \mathcal{R}$ is at most $2r - 1$.

Theorem 3.3.1. *For a SD scheme with r revoked users, the message expansion is $4r - 2$.*

(2) **Storage complexity at the server end:**

The broadcast center (server) has to store all the secret keys of the system. The following theorem shows the data complexity at the server end under the assumption that the keys are generated uniformly at random.

Theorem 3.3.2. *In a SD where the long-lived keys are generated uniformly at random and independently from each other, the server needs to store $2n(\log_2 n - 1) + 3$ keys, where n is the number of users in the system.*

Proof. The total number of long-lived keys in the system is equal to the number of subsets of $\mathcal{U}$ in S. According to the SD scheme, every node in the complete binary tree T at depth k can form a subset with each one of its ancestors, and there are k ancestors. So,

$$\sum_{k=0}^{\log_2 n} (k * 2^k) = (\log_2 n - 1) * 2^{\log_2 n + 1} + 2 = 2n(\log_2 n - 1) + 2$$

Plus $\gamma(0, 0)$, the server needs to store $2n(\log_2 n - 1) + 3$ keys. □

Although this is a vast improvement over the worst case presented in scheme Trivial B, it is still not good enough for a system the size of AACS. For example, if T is of height 31 and each long-lived key has size 128 bits, the server must store more than a terabyte of long-lived keys securely.

(3) **Storage complexity at the client end:**

We now determine how many long-lived keys a user, u, is required to store, that is, how many subset-differences is u a member of. Suppose the tree T containing u has n leaves. We start by considering the subtree of T of height h that contains u. There are $2^{h+1} - 1$ nodes in this subtree and $h + 1$ nodes

on the path between u and the subtree's root, i. Only these nodes cannot be used to form subset differences with i, otherwise u would be excluded. Hence, $2^{h+1} - 1 - (h+1)$ subset differences containing u can be formed with i in this subtree. u is in $\log_2 n$ subtrees in T, so the total number of subset differences containing u is

$$\begin{aligned} & \sum_{h=1}^{\log_2 n} (2^{h+1} - 1 - (h+1)) + 1 \\ = & \sum_{h=1}^{\log_2 n} 2^{h+1} - \sum_{h=1}^{\log_2 n} h - \sum_{h=1}^{\log_2 n} 2 + 1 \\ = & (2^{\log_2 n+2} - 4) - \frac{\log_2 n(\log_2 n + 1)}{2} - 2\log_2 n + 1 \\ = & 4n - \frac{\log_2^2 n}{2} - \frac{5\log_2 n}{2} - 3. \end{aligned}$$

When n is large, the above number is roughly equal to $4n$. With random key assignment, each user has to store as many long-lived keys as the number of subset differences he/she is in, which means that a user has to store about $4n$ keys.

Theorem 3.3.3. *In a SD scheme with random key assignment, the data complexity at the client end is* $4n - \log_2^2 n/2 - 5\log_2 n/2 - 3$.

This is undesirable for larger systems. For example, if a system has 2^{31} users and each long-lived key has size 128 bits, then each user has to store over a hundred gigabytes of long-lived keys securely.

In the next section, we introduce a key assignment strategy where the keys are generated from some secret information. This technique greatly reduces the key storage requirement at both the server end and the client end.

Computational key assignment

Up until this point, all long-lived keys in the subset difference revocation scheme have been chosen uniformly at random and independently from each other. As we

have seen, this key assignment strategy induces great storage overhead for both the broadcast center and the user. To reduce storage complexity we can instead calculate the long-lived keys from some secret information.

The idea is that instead of giving the user long-lived keys, a number of "secret seeds" are given to the user, such that many long-lived keys can be calculated from a single seed. For each pair of nodes (i, j) such that j is a descendant of i, there is a seed, $SEED_{i,j}$ associated with it. The long-lived key $L_{i,j}$ for $\gamma(i, j)$ as well as all the seeds for node pairs (i, j'), where j' is a descendant of j, can be calculated from $SEED_{i,j}$. To implement this idea, a pseudo-random sequence generator is used.

Let function, $G : \{0,1\}^a \rightarrow \{0,1\}^b$, where $b = 3a$, be a pseudo-random sequence generator such that the size of the output is 3 times the size of its input. A necessary condition for G to be a pseudo-random sequence generator is that given a randomly chosen input string, no polynomial-time adversary can distinguish the output of G from a random string of the same length. G takes a seed as its input, and outputs

$$G(SEED_{i,j}) = SEED_{i,2j}||L_{i,j}||SEED_{i,2j+1},$$

where "||" is the concatenation operator, $L_{i,j}$ is the long-lived key for $\gamma(i, j)$, and the two newly generated seeds are for $(i, 2j)$ and $(i, 2j + 1)$. Since we are using breadth-first labeling for the nodes, $2j$ and $2j + 1$ are the two children of j. Figure 3.4 gives a pictorial example to show that many secret seeds, and consequently many long-lived keys, can be generated from a single seed.

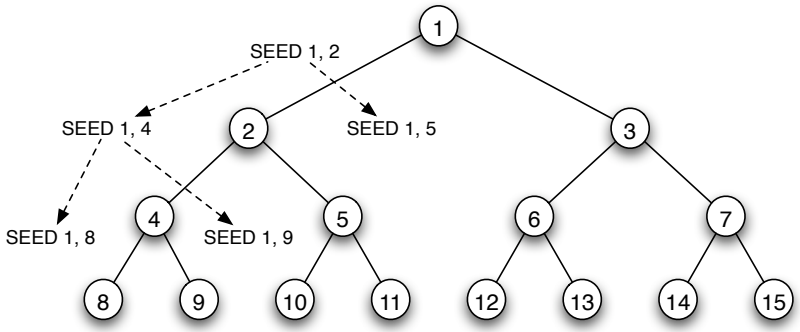

Figure 3.4: Many secret seeds can be generated from one seed.

A user u is given the seeds $SEED_{i,j}$, for all i such that i is on the path from u to the root (u excluded). The j's are the nodes that "hang off" the path from i to u (u excluded). With this setup, all long-lived keys required by u can be generated from the set of seeds. For i at height h, there are h different j's "hanging off" the path from i to u, hence the total number of seeds held by u is

$$\sum_{i=1}^{\log_2 n} i = \frac{((\log_2 n) + 1)\log_2 n}{2} = \frac{1}{2}\log_2^2 n + \frac{1}{2}\log_2 n.$$

We also need to include the key for the case when there is no revocation, which we take to be the long-lived key $L_{0,0}$. Thus, the total number of secret values a user must store is

$$\frac{1}{2}\log_2^2 n + \frac{1}{2}\log_2 n + 1.$$

Theorem 3.3.4. *In a SD scheme with keys calculated from some secret information, the data complexity at the client end is* $(\log_2^2 n + \log_2 n)/2 + 1$.

This is a vast improvement over $O(n)$ of random key assignment. For the same example from the last section where a system has 2^{31} users and each seed/long-lived key has size 128 bits, u is only required to store around 8 kilobytes (instead of more than one hundred gigabytes) of key information securely using the computational key assignment approach. Of course the catch is that this approach trades information-theoretic security for computational security.

The storage requirement for the broadcast center is greatly reduced as well. The broadcast center only requires $SEED_{i,i}$ for each non-leaf node i in order to generate all the seeds required by the users. These seeds are called *setup-seeds*. So, for a tree with n leaves, the broadcast center must store $n-1$ setup-seeds. Computational key assignment reduces the storage requirement for the server to $O(n)$ from $O(n \log n)$ required by random key assignment. Continuing with the same example in Section 3.3.3, the server is required to store about 32 gigabytes (instead of over 1 terabyte) of secret key information for computational key assignment. Although this is an improvement, storing 32 gigabytes of data securely still places a great deal of stress

on the server end. Further improvements are made possible by generating the setup-seeds in a non-information-theoretic manner. For example, instead of storing each setup-seed, the server can generate it for any internal node using a keyed pseudo-random function which takes as an input the name of the node. This reduces the storage complexity at the server end to O(1), considering that only the key for the pseudo-random function must be stored.

In the next section we discuss how the subset difference key revocation scheme is used in AACS to recover the necessary information to decrypt the encrypted content in the pre-recorded setting.

3.4 AACS Broadcast Encryption for Pre-recorded Media

3.4.1 The Basic Procedure

Pre-recorded media refers to any media that is distributed with content pre-loaded, although in current implementations it generally refers to HD-DVD and Blu-ray discs pre-recorded with video, games, or software. On the other hand, recordable media usually refers to the set of blank HD-DVD and Blu-ray discs on which consumers can record their own data. For ease of explanation, we assume that optical discs are the storage medium being used.

Content encryption is implemented by the replicator, while decryption is performed by the playback device at the client end. Figure 3.5 shows an overview of the content encryption and decryption processes. Content encryption is performed by the licensed replicator and content decryption is carried out by the playback device. Actually, the figure is a simplified version that omits the processing of *sequence keys*. Sequence keys, a special set of keys used for tracing compromised devices, will be studied in detail in Chapter 5. The simplification is made for the

purpose of increasing the comprehensibility of the basic AACS broadcast encryption scheme.

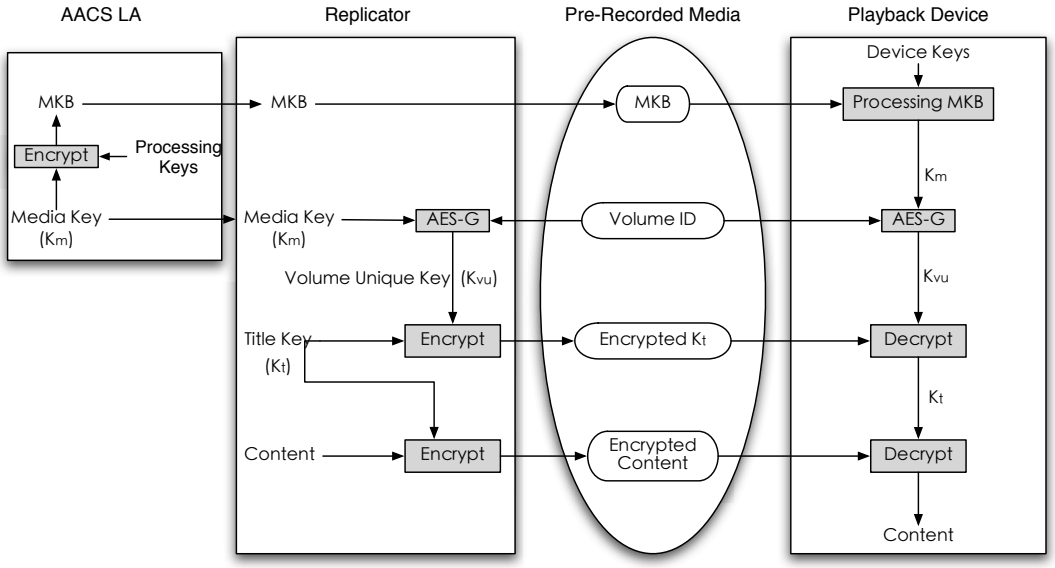

Figure 3.5: Simplified encryption and decryption for pre-recorded media

Before explaining the procedures for content encryption and decryption, we first introduce some terms and cryptographic elements used in AACS.

Title: The content owner, for example, a Hollywood movie studio, provides the content to a licensed replicator in the form of one or more *titles*. For example, a movie, such as "King Kong", can be a title.

Title Key ($\mathbf{K_t}$): The replicator randomly generates a secret 128-bit *title key* for each title. At the replicator's discretion, the title key can be used to encrypt all instances of a given title, or different title keys may be used for different instances.

Volume ID: The replicator also randomly generates a 128-bit identifier for each title, called the *volume identifier* (volume ID), which is then burned onto the disc in a special area called the "burst cutting area". This special area on either pre-recorded or recordable discs is read-only for consumer recorders. The purpose of the volume ID is to protect against "bit-by-bit copying" of

protected content. As with the title key, the same volume ID can be used for all instances of a given title, or different volume IDs may be used for different instances.

Media Key ($\mathbf{K_m}$): *Media key* is provided by the AACS Licensing Administrator (AACS LA), and is used together with the volume ID to compute the volume unique key. As with most AACS keys, the media key has length 128 bits.

Volume Unique Key ($\mathbf{K_{vu}}$): *Volume unique key* is used to encrypt the title key. It is the output of a one-way function, AES-G, which takes as inputs the volume ID and the media key.

AES: *Advanced Encryption Standard* is the building block for data encryption and decryption in AACS. The content is encrypted and decrypted using AES in CBC mode. The various keys are encrypted and decrypted using AES in ECB mode. Since most of the keys are of size 128 bits, one AES encryption or decryption is sufficient for one key encryption or decryption.

AES-G: *AES-G* is an AES-based one-way function that takes two inputs of length 128 bits, and produces an output of size 128 bits. The AES-G function is depicted in Figure 3.6. x_1 is used as a key to encrypt x_2.

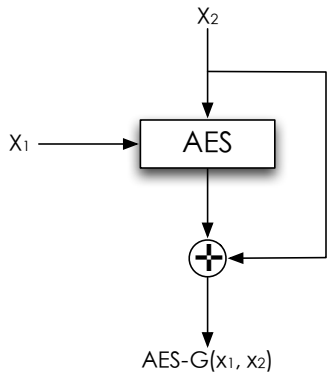

Figure 3.6: AES-based one-way function AES-G

Device Key: *Device keys* serve the same purpose as seeds in the subset difference revocation scheme. Each has length 128 bits and is stored in the playback

device.

Processing Key: *Processing keys* are the equivalent of long-lived keys in the subset difference revocation scheme. Each has length 128 bits and can be derived from the device key. Processing keys are used to encrypt the media key.

Media Key Block (MKB): *Media Key Block* is provided by the AACS LA, and contains lists of information about revoked hosts and drives. In addition, the media key is encrypted with one or more processing keys and placed in the media key block along with the indices of the processing keys used in the encryption (these together are equivalent to the header of the broadcast in the subset difference revocation scheme).

The following descriptions of content encryption and decryption correspond to Figure 3.5.

Content Encryption:

1. The content owner gives the content to the licensed replicator, who randomly generates a secret title key for the content. The content is then encrypted with the title key using AES in CBC mode.

2. For each title (or a set of titles), the AACS LA gives a MKB and a secret media key to the replicator. The media key is encrypted by a set of processing keys. The encrypted media keys are included in the MKB, and only privileged users are able to decrypt one of them to recover the media key.

3. The replicator randomly generates a volume ID for the content. It then uses the volume ID together with the media key to calculate the volume unique key using the AES-G function (media key is x_1, and volume ID is x_2 in Figure 3.6). The title key is encrypted with the volume unique key using AES.

4. Finally, the replicator places the MKB, volume ID, encrypted title key, and encrypted content onto the disc, along with several other pieces of information

not directly related to content encryption. Note that the volume ID is placed inside the "burst cutting area" on the disc and is read-only for all consumer recorders.

Content Decryption:

1. If the playback device is not revoked, then due to the subset difference revocation scheme it should be able to use its device keys to derive a processing key which can decrypt one of the encrypted media keys recorded in the MKB.

2. The playback device then reads the volume ID off the disc and inputs it together with the media key into the AES-G function to calculate the volume unique key.

3. Upon obtaining the volume unique key, the title key is recovered by decrypting the accompanying encrypted title key.

4. Finally, the playback device decrypts the encrypted content using the title key and plays the content.

As indicated in the technical specifications [8], each device stores 253 device keys. Recall that a device key is the same as a seed in the subset difference revocation scheme. If a user stores 253 seeds, it implies that the height of the complete binary tree is 22, according to our analysis in Section 3.3.3. However, we have not yet found any authoritative information regarding the size of the complete binary tree used in AACS (it is vaguely mentioned in a paper [82] written by one of the designers of AACS).

3.4.2 Content Revocation

There is a fairly straightforward authentication from the disc to the playback device to achieve content revocation. The content revocation mechanism adds a layer of control over the content. It enables the AACS LA to revoke certain titles, prevent

malicious alteration of legitimate content, and disable playback of unauthorized titles. Figure 3.7 shows a diagram of the content revocation mechanism, and a corresponding description is provided below.

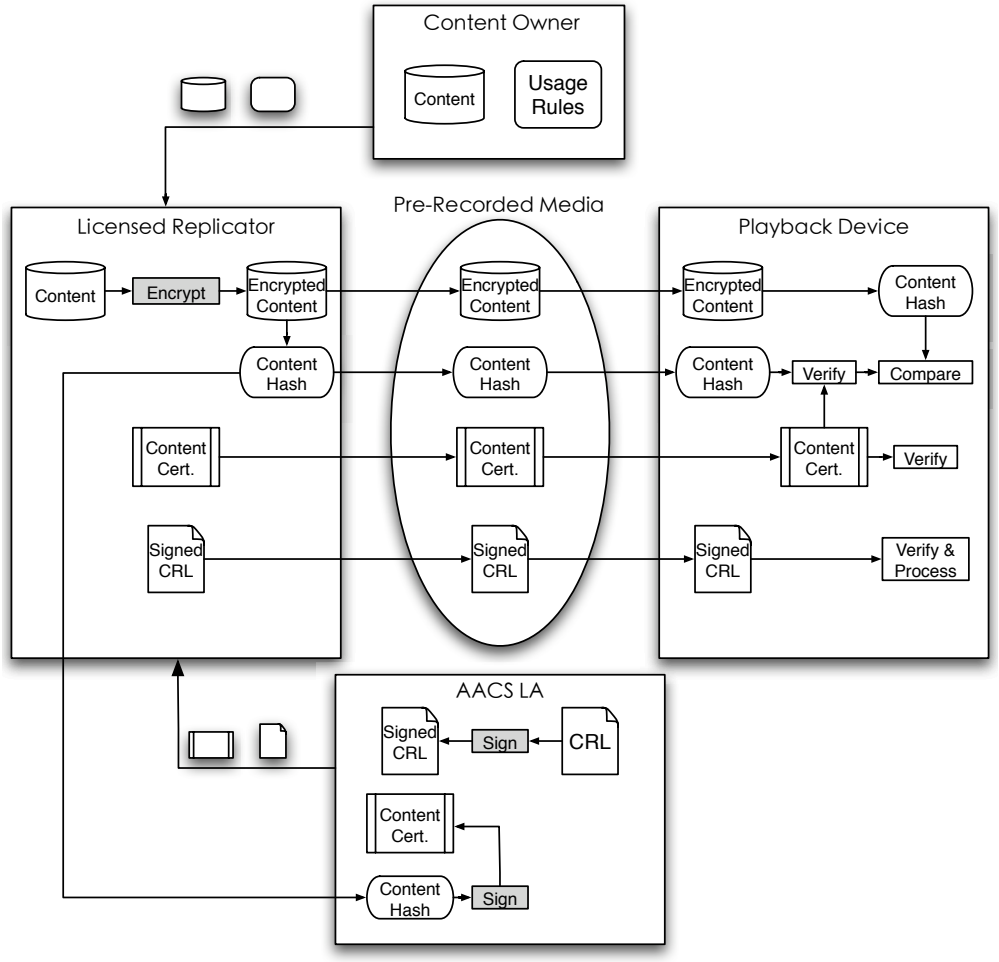

Figure 3.7: Content revocation of pre-recorded media

The owner of the title that is to be protected provides the title and the usage rules to a licensed replicator. The licensed replicator performs the content encryption described in the previous section and then calculates a hash of the encrypted content. The hash is then sent to the AACS LA. The licensed replicator cryptographically binds the title key for the title to those usage rules to prevent any malicious alteration of those usage rules (omitted in Figure 3.7).

Upon receiving the content hash, the AACS LA signs it using an AACS LA

public key to produce a *content certificate* and sends the certificate back to the licensed replicator. Along with the certificate, the AACS LA also sends back a signed up-to-date *content revocation list* (CRL). The CRL contains a list of revoked titles that have been signed and issued valid certificates but should no longer be accessed by a compliant player. The licensed replicator puts the CRL, content hash, and content certificate onto the pre-recorded disc.

When the pre-recorded disc is inserted into a compliant playback device, the device verifies the content certificate and the CRL. If any of the two verifications fail, the device should stop processing the disc. The device calculates a hash of the encrypted content the same way as the replicator did (This is according to the specification [9], however, specific implementations might not choose to calculate a hash on the entire encrypted content as it is very time consuming to do so). This hash is compared with the content hash stored on the disc. If they are not the same, the device can conclude that the content has been altered, so further processing of the disc should be aborted. Each compliant device should have at least 128 kilobytes of non-volatile storage for the purpose of storing the most recent CRL. If the CRL on the disc is newer, the device updates its CRL to the one on the disc. The device checks whether the content inserted in the device is listed in the CRL. If yes, the device should terminate accessing the content.

3.4.3 Real-World Security of AACS

It would seem that the encryption protocol used by AACS is robust, but this begs the following question: Why have so many HD-DVD and Blu-ray movies been pirated? The reality is that attackers have been able to recover various keys used in the encrypting process of the content. Since December 27, 2006, attackers have managed to locate the title key for any title by taking memory dumps from Microsoft Windows-based HD-DVD/Blu-ray software decoders *WinDVD* and *PowerDVD* [87]. Once a title key is compromised, anyone who has the corresponding title can decrypt the encrypted content freely, bypassing the usual AACS key re-

covery. On February 11, 2007, attackers recovered two processing keys for both HD-DVD and Blu-ray discs in another memory dump of WinDVD [16]. Recall that the processing key is equivalent to the long-lived key in the subset difference revocation scheme. At the time of the compromise, no devices had been revoked, so all media keys for all HD-DVD and Blu-ray discs were encrypted with these two compromised processing keys. Knowledge of these keys allows all discs released prior to revocation to be decrypted. On February 24, 2007, attackers managed to extract some of the device keys stored in WinDVD [112].

On April 16, 2007, the AACS LA announced that the device keys for the software decoders were finally revoked [11]. The AACS LA has blamed poor software implementation for the compromise, rather than a weakness in the system itself. Nevertheless, the leaking of just two processing keys has allowed all HD-DVD and Blu-ray movies released before April 2007 (the first HD-DVD and Blu-ray titles were released in April 2006 and June 2006 respectively) to be copied freely.

Unfortunately for the AACS LA and the movie studios, on May 23, 2007 a new processing key was posted on the web, and this key was not revoked until September 7 [11]. The reason why it took so long for the AACS LA to revoke this key is due to the fact that manufacturers need to redesign their products in order not to be exploited again by the attacker. If the manufacturer merely replaces the compromised keys with the new ones, the attacker can just re-apply whatever techniques they used to extract the previous keys to obtain the new keys. In order to have enough time to update their product, the AACS Interim Adopter Agreement [3] says that manufacturers can receive a 90-day notice before their keys get revoked.

The bad news for the AACS LA does not end there. SlySoft is an Antigua-based company whose product *AnyDVD HD* can effectively circumvent AACS and pirate almost all the HD-DVD and Blu-ray titles on the market published to date. AnyDVD HD is a Mircosoft Windows-based driver that works in the background and provides decryption to high definition media. It thus can cooperate with *CloneDVD*,

which is another product of SlySoft, to produce un-copy-protected high definition movies. Unlike DeCSS, mentioned in Section 2.2, AnyDVD HD simply uses leaked device/processing keys that have not yet been revoked to decrypt the HD-DVD and Blu-ray titles. The AACS LA is probably still in the process of trying to figure out what those compromised keys are. Again, the AACS scheme itself has not been broken, but attackers are still able to extract the device/processing keys.

If the key revocation cycle was always longer than the time needed by the attacker to extract the secret keys, then it would constantly be a losing battle for the AACS LA. Actually, if it took only a few months for the attacker to extract the new keys, the effectiveness of AACS to protect intellectual property would be greatly diminished regardless of the length of the key revocation cycle, because it would take only (at most) a few months for the pirated movie to appear after the authentic one was released. A natural question regarding this issue is: Can we reduce the ability of a set of leaked device/processing keys to pirate movies published before the revocation of the compromised keys? One way of achieving this goal is to have the scheme to provide *perfect forward secrecy* [53]. A broadcast encryption scheme provides perfect forward secrecy if disclosure of a set of device keys does not compromise the title keys of the movies published earlier.

The title key is a session key. One session key is responsible for one particular session, which in the case of AACS is represented by a movie distribution. Compromising a session key should only affect the secrecy of its corresponding session, and the secrecy of other sessions should remain intact as long as their session keys are not compromised. Device/Processing keys are long-lived keys because they are used for multiple sessions. Due to the stateless nature of the receiver in AACS, long-lived keys are used almost throughout the entire lifetime of the device. It is desirable that the compromising of a session key should not reveal information about the long-lived key. An attack model is called *known session key attack* if the attacker learns the value of a particular session key and uses it to do whatever he/she can. If the attacker knows the long-lived key, then the attacker can carry

out a *known long-lived key attack*.

If a session key is obtained by the attacker, he/she can only pirate the movie encrypted by that session key. However, if the attacker gets the long-lived key, he/she can pirate every movie on the market. Perfect forward secrecy demands that, even if the long-lived key is compromised, the attacker cannot learn the values of previous session keys, i.e., he/she cannot pirate previously published movies.

Perfect forward secrecy in the context of public-key broadcast encryption has recently become an active research topic. A few forward-secure public-key broadcast encryption schemes have been proposed ([42], [120]). They allow each user to update his/her private key periodically while keeping the public key unchanged. To broadcast a message to a user, the content is encrypted by the public key and the time period corresponding to the user's private key. Even if the adversary learns the private key of some user at time period t, messages encrypted during all times prior to t remain secret. However, the user also loses the ability to decrypt any old encrypted messages broadcasted prior to t. Although these schemes can be applied to the stateless model, they are not suitable for content distribution systems such as AACS where the user should have the ability to play back any movie published in any time period.

In addition to updating the long-lived key periodically, perfect forward secrecy can also be realized by using Diffie-Hellman key agreement style protocols where the long-lived key is not directly involved in providing confidentiality to the session key (instead it is used for other purposes such as identity authentication [92]). However, this approach does not work for revocation schemes in general because the long-lived key is designed to provide confidentiality to the session keys.

The task of incorporating the feature of perfect forward secrecy into a revocation scheme for multi-media content distribution could be hard. The limitation is inherent to the content distribution model. If a licensed device is compromised without the AACS LA knowing about it, the leaked device/processing keys could be used to pirate any movie on the market due to the fact that a legitimate player

can decrypt any movie published at any time. Even if the AACS LA knows which keys are compromised, there is still very little it can do. Since the movie discs are already on the market, which means that the AACS LA cannot do any modification to the encrypted message, the keys that could decrypt them before will be able to decrypt them after they are compromised.

There might be other ways to limit piracy in the case of a known long-lived key attack. On June 28, 2007, BD+ Technologies, LLC, announced the release of a new technology, BD+, which is a virtual machine-based content protection technology created exclusively for the Blu-ray disc format [26]. This technology works with AACS to provide another layer of defense against piracy. Allegedly, a certain analyst claims that BD+, (unlike AACS, which suffered a partial hack) will not likely be breached for 10 years, and if so, the damage would affect only one movie and one player [61]. Since the BD+ specifications are confidential, most of the details of the scheme are unknown. However, it is certainly not true that this technology will stand unbreakable for 10 years, considering Slysoft released an update of its AnyDVD HD software on November 7, 2007 and the new version can crack any BD+ protected disc [19]. Since the BD+ protected movies can be copied freely, the claim of limiting the damage to one movie and one player also failed. At this stage, limiting piracy when a device's long-lived keys are compromised is still an open problem.

3.5 Suggested Improvements to AACS Broadcast Encryption

In this section, we suggest some improvements to AACS broadcast encryption based on some existing variations of the SD scheme. The first suggested improvement reduces the client storage requirement while keeping other factors of the system essentially the same. The second suggested improvement converts the original symmetric-key SD scheme into a public-key SD scheme which enables everyone

to broadcast messages and eliminates the need for the server to store secret information.

3.5.1 Generalized Subset Difference Revocation Scheme

As shown in Section 3.3.3, the user is required to store $O(\log^2 n)$ keys securely in the subset difference revocation scheme. Reducing client storage complexity is the most effective way to reduce the cost of the system, because the fewer keys the device needs to store securely, the cheaper it is to implement the hardware. Halevy and Shamir [63] have generalized the subset difference revocation scheme to achieve extremely close to $O(\log n)$ keys per user while retaining other complexities.

The generalized subset difference scheme is termed *layered subset difference* (LSD) scheme. There are two versions of the LSD scheme: the basic version and the generalized version. The basic version is a very practical scheme that achieves client storage complexity $O(\log^{3/2} n)$ and maintains message expansion $O(r)$. The generalized version gets essentially $O(\log n)$ keys per user with anything else being almost equal; however, it is theoretical rather than practical. Let us now study the basic scheme in detail.

The basic scheme

Let $\mathcal{U} = \{u_1, u_2, \ldots, u_n\}$ be a set of n users where $n = 2^h$, such that each user corresponds to a leaf node in a complete binary tree, T, of height h. For ease of explanation, let us assume that $\sqrt{h}$ is an integer. We can partition the tree into layers with each layer having $\sqrt{h}$ levels. Hence, we have a total of $\sqrt{h}$ layers for T. Levels at depth $\alpha\sqrt{h}$, $\alpha \in \{0, 1, \ldots, \sqrt{h}\}$ (i.e., a boundary of a layer), are called special levels. Let $D(i)$ denote the depth of node i, so $D(i) \in \{0, 1, \ldots, h\}$, for all $i \in T$.

Definition 3.5.1. A subset difference $\gamma(i,\ j)$ is *useful* if it is not empty, and i is at a special level, that is, $\sqrt{h}|D(i)$, or i and j belong to the same layer, that is,

$\alpha\sqrt{h} < D(i) < D(j) \leq (\alpha+1)\sqrt{h},\ \alpha \in \{0, 1, \ldots, \sqrt{h}-1\}$.

Lemma 3.5.1. *Any nonempty difference set $\gamma(i,\ j)$ is either a useful set or the disjoint union of two useful sets.*

Proof. If $\gamma(i,\ j)$ is useful, then this is trivial. If $\gamma(i,\ j)$ is not useful, then $\alpha\sqrt{h} < D(i) < (\alpha+1)\sqrt{h} < D(j),$ for some $\alpha \in \{0, 1, \ldots, \sqrt{\mathrm{h}}-1\}$. Let k be the node on the path from i to j that is sitting on level $(\alpha+1)\sqrt{h}$. $\gamma(i,\ j) = \gamma(i,\ k) \cup \gamma(k,\ j)$, and both $\gamma(i,\ k)$ and $\gamma(k,\ j)$ are useful. □

Instead of assigning a long-lived key to each subset difference as we did in the subset difference revocation system, we assign it a long-lived key only if the subset difference is useful. Every user in this subset difference has a copy of the secret long-lived key.

Now let us calculate how many keys a user has to store. The number of keys is determined by the number of useful subset differences, $\gamma(i,\ j)$, that include the user. i is on the path from the user to the root of T, and j is a node that "hangs off" the path from i to the user. i and j have to satisfy either of the two conditions: $\sqrt{h}|D(i)$, or $\alpha\sqrt{h} < D(i) < D(j) \leq (\alpha+1)\sqrt{h},\ \alpha \in \{0, 1, \ldots, \sqrt{h}-1\}$. We count the total number of useful sets that a user is in by counting these two different types of useful sets separately:

1. $\sqrt{h}|D(i)$: i is at a special level $\alpha\sqrt{h}$, for some $\alpha \in \{0, 1, \ldots, \sqrt{h}-1\}$. For every one of the $h-\alpha\sqrt{h}$ levels below, there is a unique j at that level hanging off the path from i to the user that can form a useful set with i. Hence, there are a total of

$$\sum_{\alpha=0}^{\sqrt{h}}(h-\alpha\sqrt{h}) = (\sqrt{h}+1)h - \frac{1}{2}(\sqrt{h}+1)h = \frac{1}{2}h(\sqrt{h}+1)$$

   useful sets of this type containing the user.

2. $\alpha\sqrt{h} < D(i) < D(j) \leq (\alpha+1)\sqrt{h},\ \alpha \in \{0, 1, \ldots, \sqrt{h}-1\}$: i and j are within the same layer. The number of useful sets within a layer that contain the user

is

$$\sum_{i=0}^{\sqrt{h}-1} i = \frac{\sqrt{h}(\sqrt{h}-1)}{2}.$$

T has $\sqrt{h}$ layers, so the user is in $h(\sqrt{h}-1)/2$ useful sets of this type.

Summing up the two disjoint counts, the user needs to store

$$\frac{1}{2}h(\sqrt{h}+1) + \frac{1}{2}h(\sqrt{h}-1) = h^{3/2}$$

long-lived keys. Since $h = \log_2 n$, the storage complexity for the client is $\log_2^{3/2} n$.

To find the subset-cover that covers every non-revoked user in this scheme, the center uses algorithm 1 in Section 3.3.2 to find a corresponding set of subset differences, and replaces any of them that is not useful by two useful sets. In Section 3.3.3, we showed that the size of the subset-cover for $\mathcal{U} \setminus \mathcal{R}$ is at most $2r-1$ in the subset difference revocation scheme. Hence, the header length in this scheme is at most $8r-4$. Therefore, the complexity of the message expansion is still $O(r)$, although with a bigger factor.

It is not hard to see the motivation for creating this scheme: to reduce the number of subsets of which the user is a member at a cost of increasing the message expansion. Fortunately, the gain outperforms the cost, because the storage space is much cheaper on a high-definition media disc than on hardware such as EEPROM. The basic scheme splits a subset into at most two subsets. Can we split a subset into more subsets? Yes, and that is the main idea of the generalized LSD scheme. In the generalized scheme, a subset can be replaced by $d > 2$ subsets, that is, if nodes $i, k_1, k_2, \ldots, k_{d-1}, j$ occur in this order on a path, a subset $\gamma(i,j)$ can be represented by $\gamma(i,\ k_1) \cup \gamma(k_1,\ k_2) \cup \ldots \cup \gamma(k_{d-1},\ j)$. This increases the upper bound on the header size to $d(2r-1)$, but also decreases the client storage complexity to $O(log^{1+\frac{1}{d}} n)$. For more detail about the generalized LSD scheme, please refer to [63]. Halevy and Shamir point out that the generalized LSD scheme is mostly of theoretical interest, and it requires an astronomical number of users in order to see the advantages of it over the basic LSD scheme. Hence, we will not consider the generalized version when we apply the LSD scheme to AACS.

Applying the basic LSD scheme to AACS

The basic LSD scheme can be applied to AACS in a way that is almost identical to the implementation of the subset difference revocation scheme. One of the few things that needs to be changed is the device key distribution. For each user, the basic LSD scheme determines a set of subsets containing the user. Each subset corresponds to a device key and all the users in this subset are given this key. According to the analysis from the last section, the LSD scheme enables the user to store fewer device keys, as compared to the subset difference revocation scheme.

Since there are already so many devices on the market, the new devices based on the basic LSD scheme have to be compatible with the old devices. To achieve this goal, we can simply take the union of the message header corresponding to the subset difference revocation scheme, which has message expansion $4r - 2$ and the message header corresponding to the LSD scheme, which has message expansion $8r - 4$, and put the union on the disc. The message expansion for the new header is no bigger than $12r - 6$, which still gives us overall complexity $O(r)$. Legitimate devices implementing the SD scheme can use their device key to decrypt the encrypted session key stored in the SD part of the header, and others implementing the LSD scheme can recover the encrypted session key stored in the LSD part of the header.

3.5.2 Public-key SD Scheme

The subset difference revocation scheme described in Section 3.3.2 is built on symmetric-key cryptographic primitives. There are some limitations with symmetric-key broadcast encryption schemes, largely due to the fact that the broadcast center needs to store all the secret keys in the system and that they cannot be easily made available to other entities. Under such circumstances, no entity except the broadcast center can broadcast messages to a group of users, because unlicensed parties simply do not have the corresponding secret keys. It also makes the broadcast

center a single point of failure in the case of a security breach.

On the other hand, a public-key broadcast encryption scheme enjoys more freedom than its symmetric-key counterpart. The message is encrypted with public keys, so everyone can make a broadcast. The encrypted message is decrypted using the private keys, which are distributed among the users. The problem of a single point of failure is alleviated since there is no need for an entity to store all the private keys in the system. (However, this problem is not entirely solved as we will discuss later in Section 3.5.2.)

One interesting problem observed by D'Arco and Stinson is that the licensed distributor might not be trustworthy [52]. The licensed distributor receives content from the content provider in the clear, and technically it can illegally distribute the content without being easily accused by the licensing entity. In the presence of such rogue distributors, the broadcast encryption system need to be designed so that the content is protected from being illegally distributed by the distributor without sacrificing the content availability to privileged users. We do not study this topic in this thesis. D'Arco and Stinson have suggested a fault tolerant and distributed broadcast encryption as a solution to this problem [52].

As already mentioned by Naor et al. in their original paper [89], the general subset-cover framework can be trivially adapted to the public-key setting, by having each long-lived key L_j for a subset S_j be replaced by a pair of public/private keys (Pub_j, Pri_j). However, this results in a huge number of private keys for the system. In this section, we will show how to convert the original SD scheme into a public-key based revocation scheme using *hierarchical identity-based encryption* (HIBE). The resulting public-key SD scheme resembles closely the original SD scheme under the computational key assignment setting, and hence enjoys similar message expansion complexity and storage complexities at the server and the client ends.

HIBE is a powerful generalization of *identity-based encryption* (IBE). We will first study IBE and HIBE, and then show the construction of the public-key SD scheme followed by an analysis of the newly built public-key SD scheme.

Identity-based encryption

One of the challenges in public-key cryptography is ensuring the authenticity of public keys. In the traditional model where public keys are random strings, *public-key infrastructures* (PKI) have to be established to facilitate the verification of public keys [101, Chapter 12]. However, PKI does not provide a perfect solution as it introduces other risks and inconveniences [56]. An alternative approach is to use some unique information about the identity of the user which is known to everyone, such as his/her email address, as his/her public key. This approach is therefore termed identity-based encryption. IBE effectively eliminates the need to verify the authenticity of users' public keys using certificates.

The idea of identity-based encryption was proposed by Shamir [96] in 1984; however, practical schemes were not found until recently with the work of Boneh and Franklin [34] in 2001. Since then, identity-based cryptography has received much attention. Baek et al. [18] present a good survey on this topic.

Generally speaking, the user's public key in IBE can be an arbitrary string chosen by the user. Since the main benefit of the public key in IBE is the ability to identify the user, it is always chosen as an identifier of the user. A trusted entity known as *private key generator* (PKG) is responsible for calculating the corresponding private key from a public key. The entire identity-based encryption scheme can be modeled by a combination of four steps, where each step is a polynomial-time algorithm:

1. **Setup(λ):** A probabilistic algorithm used by the PKG to initialize the global parameters of the system. Given a security parameter λ, Setup generates a set of system global parameters *params* as the global public key, and it usually remains constant for a long period. This algorithm also generates a master secret key *master-key* which is kept secret by the PKG.

2. **Extract(**ID**,** params**,** master-key**):** An algorithm used by the PKG to derive the corresponding private key from a public key. A user authenticates him-

self/herself to the PKG and provides it with his/her ID (public key). Extract takes ID, params, and master-key as input, and returns to the user a private key d capable of decrypting ciphertexts generated by using his/her ID.

3. **Encrypt(**ID**,** params**,** M**):** A probabilistic algorithm used to encrypt a message M to a user with his/her identifier ID. Encrypt takes as input ID, params, and M and generates a ciphertext C to user ID.

4. **Decrypt(**ID**,** params**,** d**,** C**):** A deterministic algorithm used to decrypt a ciphertext C intended for a user with identifier ID. Decrypt takes as input ID, params, the user's private key d, and C and outputs the plaintext M.

It is required that the system must be correct, namely that for any set of system public parameters params output by Setup, any ID and the corresponding private key d, and any message M it must be that:

$$\textbf{Decrypt}(\text{params, ID, } d, \textbf{Encrypt}(\text{params, ID, } M)) = M.$$

The Boneh and Franklin IBE scheme supports semantic security against adaptive chosen-ciphertext attack while allowing the attacker to choose the challenge identity adaptively (IND-ID-CCA-2) [34].

Hierarchical identity-based encryption

IBE is not efficient enough for large networks, because the PKG becomes a bottleneck. It has to authenticate every user and establish secure channels to transmit private keys to the user. Hierarchical identity-based encryption schemes were proposed to alleviate the workload of a single PKG by delegating private key generation and identity authentication to more PKGs.

HIBE organizes the users into a hierarchy (pictorially represented by a tree). A single PKG at the top level of the hierarchy is called the root PKG. The root PKG only needs to generate private keys for the users in the second level, which is directly below it. The users on the second level in turn act as PKGs to generate private

keys for the users one level below them, and this continues downward through the entire hierarchy. Identity authentications and private key transmissions can be done locally.

In 2002, Gentry and Silverberg [62] constructed an HIBE based on the *Bilinear Diffie-Hellman* (BDH) assumption in the random oracle model. Subsequently, Boneh and Boyen [32] developed another HIBE scheme based on BDH without random oracles. In Section 3.5.2, we will present an HIBE scheme (called the BBG scheme) proposed by Boneh, Boyen, and Goh [33]. This scheme will be employed in the development of a public-key SD scheme in Section 3.5.2.

Similar to IBE, a HIBE scheme can be modeled by four components: Root-Setup, Extract, Encrypt, and Decrypt. Since a HIBE scheme organizes users into a hierarchy, user IDs are vectors. A vector of dimension k, $(I_1, \ldots, I_k)$, represents an identity at depth k.

1. **RootSetup(λ):** A probabilistic algorithm used by the root PKG to initialize the global parameters of the system. RootSetup takes as input a security parameter λ and generates the system's global public key params. It also generates a master secret key master-key and makes it known only to the root PKG.

2. **Extract($\mathrm{ID}_{|k}$, params, $d_{ID|k-1}$):** A probabilistic algorithm used by a PKG to derive a corresponding private key from a public key. Extract takes as input an identity $\mathrm{ID}_{|k} = (I_1, \ldots, I_k)$ at depth k, params, and the private key $d_{\mathrm{ID}|k-1}$ of the parent identity $\mathrm{ID}_{|k-1} = (I_1, \ldots, I_{k-1})$ (if the parent is the root PKG, the private key is master-key), and returns to the user with identifier $\mathrm{ID}_{|k}$ a private key $d_{\mathrm{ID}|k}$.

3. **Encrypt($\mathrm{ID}_{|k}$, params, M):** : A probabilistic algorithm used to encrypt a message M to a user with identifier $\mathrm{ID}_{|k}$. Encrypt takes as input $\mathrm{ID}_{|k}$, params, and M and generates a ciphertext C to $\mathrm{ID}_{|k}$.

4. **Decrypt**($\mathrm{ID}_{|k}$, params, $d_{\mathrm{ID}|k}$, C): A deterministic algorithm used to decrypt a ciphertext C intended for a user with identifier $\mathrm{ID}_{|k}$. Decrypt takes as input $\mathrm{ID}_{|k}$, params, $d_{\mathrm{ID}|k}$, and C and outputs the plaintext M.

As usual, the system needs to be correct. That is:

$$\textbf{Decrypt}(\text{params}, \mathrm{ID}_{|k}, d_{\mathrm{ID}|k}, \textbf{Encrypt}(\text{params}, \mathrm{ID}_{|k}, M)) = M$$

for any system public key params output by RootSetup, any $\mathrm{ID}_{|k}$ and a corresponding private key $d_{\mathrm{ID}|k}$, and any message M.

Other state-of-the-art HIBE schemes, such as the ones proposed by Sarkar and Chatterjee [94] and Waters [117], also provide semantic security against adaptive chosen-ciphertext and adaptive chosen-identity attack (IND-ID-CCA-2).

The BBG scheme

The HIBE scheme proposed by Boneh, Boyen, and Goh has some compelling properties. Most notably, the ciphertext has constant size, and the user's private key becomes shorter as his/her depth increases in the hierarchy.

Before going into the detail of the BBG scheme, let us introduce two important definitions.

Definition 3.5.2. Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two multiplicative cyclic groups of prime order p. A map $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ is called an *admissible pairing* if the following properties are satisfied:

(1) Bilinear : For all $Q, R \in \mathbb{G}_1$ and $a, b \in \mathbb{Z}_p^*$, we have $e(Q^a, R^b) = e(Q, R)^{ab}$.

(2) Non-degenerate : The map does not send all pairs in $\mathbb{G}_1 \times \mathbb{G}_1$ to the identity in $\mathbb{G}_2$.

(3) Computable : There is an efficient algorithm to compute $e(Q, R)$ for any $Q, R \in \mathbb{G}_1$. We call $\mathbb{G}_1$ a bilinear group.

Lemma 3.5.2. *An admissible pairing e is symmetric, i.e. $e(Q,R) = e(R,Q)$ for all $Q, R \in \mathbb{G}_1$.*

This lemma follows immediately from the bilinearity property and the fact that $\mathbb{G}_1$ is cyclic.

The BBG scheme makes use of an admissible bilinear pairing. The security of the BBG scheme is based on assuming the intractability of the *Bilinear Diffie-Hellman Inversion* (BDHI) problem [32, 33].

Definition 3.5.3. Let $\mathbb{G}$ be a bilinear group of order p. Let $w \in \mathbb{G}$ be a generator and $\beta \in \mathbb{Z}_p^*$. The l-th Bilinear Diffie-Hellman Inversion problem, denoted l-BDHI, is as follows:

$$l\text{-BDHI} : \text{given } (w, w^{\beta}, w^{\beta^2}, \ldots, w^{\beta^l}), \text{ compute } e(w,w)^{1/\beta}.$$

We assume that l-BDHI in $\mathbb{G}$ is intractable.

The following description of the BBG scheme is almost identical to the original one [33]. Let $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ be an admissible pairing where $\mathbb{G}_1$ and $\mathbb{G}_2$ are two multiplicative cyclic groups of prime order p. The identifiers of users at depth k are vectors of elements in $(\mathbb{Z}_p^*)^k$, written $\mathrm{ID}_{|k} = (I_1, \ldots, I_k) \in (\mathbb{Z}_p^*)^k$. We assume that the messages to be encrypted are elements in $\mathbb{G}_2$.

RootSetup: To initialize the global parameters for an HIBE of maximum depth l, select a random generator $g \in \mathbb{G}_1$ and a random $\alpha \in \mathbb{Z}_p$, and set $g_1 = g^{\alpha}$. Next, randomly pick elements $g_2, g_3, h_1, \ldots, h_l \in \mathbb{G}_1$. The system's global public key params is $(g, g_1, g_2, g_3, h_1, \ldots, h_l)$. The master secret key master-key is g_2^{α}.

Extract: To derive a private key $d_{\mathrm{ID}|k}$ for an identity $\mathrm{ID}_{|k} = (I_1, \ldots, I_k) \in (\mathbb{Z}_p^*)^k$ of depth $k \leq l$, use params, master-key, and a random element $r \in \mathbb{Z}_p$, and output

$$d_{\mathrm{ID}|k} = (g_2^{\alpha} \cdot (h_1^{I_1} \cdots h_k^{I_k} \cdot g_3)^r,\ g^r,\ h_{k+1}^r,\ \ldots,\ h_l^r) \in \mathbb{G}_1^{2+l-k}. \tag{3.1}$$

Note that given a private key for a parent identity $\mathrm{ID}_{|k-1} = (I_1, \ldots, I_{k-1}) \in (\mathbb{Z}_p^*)^{k-1}$, the algorithm can generate a private key for any one of its children. To see how this can be done, let us assume that

$$d_{\mathrm{ID}|k-1} = (g_2^{\alpha} \cdot (h_1^{I_1} \cdots h_{k-1}^{I_{k-1}} \cdot g_3)^{r'},\ g^{r'},\ h_k^{r'},\ \ldots,\ h_l^{r'}) = (a_0, a_1, b_k, \ldots, b_l)$$

is the private key for $\mathrm{ID}_{|k-1}$. To generate $d_{\mathrm{ID}|k}$, pick a random $t \in \mathbb{Z}_p$ and output

$$d_{\mathrm{ID}|k} = (a_0 \cdot b_k^{I_k} \cdot (h_1^{I_1} \cdots h_k^{I_k} \cdot g_3)^t,\ a_1 \cdot g^t,\ b_{k+1} \cdot h_{k+1}^t,\ \ldots,\ b_l \cdot h_l^t).$$

This key is essentially of the same form as (3.1) with $r = r' + t \in \mathbb{Z}_p$. Hence, the parent can act as a PKG for all of its children.

One additional observation is that $d_{\mathrm{ID}|k}$ becomes shorter as the depth k increases.

Encrypt: To encrypt a message $M \in \mathbb{G}_2$ under the public key $\mathrm{ID}_{|k} = (I_1, \ldots, I_k) \in (\mathbb{Z}_p^*)^k$, pick a random $s \in \mathbb{Z}_p$ and output

$$C = \left(e(g_1, g_2)^s \cdot M,\ g^s,\ (h_1^{I_1} \cdots h_k^{I_k} \cdot g_3)^s\right) \in \mathbb{G}_2 \times \mathbb{G}_1^2.$$

Note that $e(g_1,\ g_2)$ used for encryption can be precomputed, so that encryption does not require any pairing operations.

Decrypt: To decrypt a given ciphertext $C = (X, Y, Z)$ using a private key $d_{\mathrm{ID}|k} = (a_0, a_1, b_{k+1}, \ldots, b_l)$ corresponding to an identity $\mathrm{ID}_{|k} = (I_1, \ldots, I_k) \in (\mathbb{Z}_p^*)^k$, output

$$X \cdot e(a_1, Z)/e(Y, a_0) = M.$$

To see the correctness of the decryption, notice that:

$$\frac{e(a_1, Z)}{e(Y, a_0)} = \frac{e\left(g^r,\ (h_1^{I_1} \cdots h_k^{I_k} \cdot g_3)^s\right)}{e\left(g^s,\ g_2^{\alpha}(h_1^{I_1} \cdots h_k^{I_k} \cdot g_3)^r\right)} = \frac{1}{e(g,\ g_2)^{s\alpha}} = \frac{1}{e(g_1,\ g_2)^s}.$$

Notice that the ciphertext contains only three group elements regardless of the depth of the identity, and the decryption takes only two pairing operations.

In the BBG scheme, any legitimate private key $d_{\mathrm{ID}|k}$ can decrypt messages encrypted to $\mathrm{ID}_{|k}$.

An important property of this scheme that we will make use of when constructing our public-key SD scheme is that a PKG can decrypt any message encrypted to any of its descendants. This is possible due to the fact that a PKG can derive a private key for any one of its descendants, and any private key corresponding to that descendant's ID can decrypt messages encrypted to him/her. Actually, this property is inherent to most HIBE schemes.

This scheme is semantically secure against chosen-plaintext and selective-identity attacks (IND-sID-CPA). Modifications can be applied to make it fully HIBE secure (IND-ID-CCA-2) [33].

Construction of a public-key SD scheme

Dodis and Fazio [54] have developed a technique to extend the original SD scheme to the public-key setting. More specifically, this extension allows some HIBE schemes to be incorporated into the SD scheme in order to make it asymmetric.

At the core of this technique lies the construction of a tree from the subset difference tree in such a way that the leaves of this tree correspond to the elements in S, the set of subset differences in the system. Let us call this tree the *hierarchical tree*, because later we will see that this tree can be used to represent the hierarchy of an HIBE scheme.

The hierarchical tree is unique per system. The procedure of constructing a hierarchical tree from a subset difference tree is fairly straightforward. Given a SD scheme, let us denote T the subset difference tree, and T' the hierarchical tree. Let Root denote the root of the hierarchy tree which functions as the root PKG. The algorithm for constructing T' is shown in Algorithm 2.

For a SD scheme with n (suppose that n is a power of 2) users, there should be exactly $n-1$ nodes at level 1 of T'. These nodes correspond to all the possible

Algorithm 2: Hierarchical Tree Construction Algorithm

2.1 Root of T' is at level 0 (depth 0).

2.2 **for** *each internal node* u *in* T **do**

2.3 Add u as a child to Root;

2.4 **end**

2.5 **for** *each node* w *at level 1 of* T' **do**

2.6 Attach two children w_l and w_r, which correspond to the left and right children of w in T, to w;

2.7 **end**

2.8 **while** *current level* $\leq$ *the last level of* T' **do**

2.9 **for** *each node* w *at this level* **do**

2.10 **if** w *is labeled with the symbol* "$\perp$" **then**

2.11 w is a leaf in T', and do nothing;

2.12 **else if** w *is a leaf in* T **then**

2.13 Attach w with a single child labeled $\perp$;

2.14 **else**

2.15 Attach three children w_l, w_r, and $\perp$ to w (w_l and w_r correspond to the left and right children of w in T);

2.16 **end**

2.17 **end**

2.18 Go down one level;

2.19 **end**

primary nodes of a generic subset difference of the system. Each leaf node $\perp$ in T' corresponds to a subset difference $\gamma(i, j)$ in S, where i is the node directly under Root on the path from $\perp$ to Root, and j is the parent of $\perp$.

Let us use an example to better illustrate the result of this algorithm. Suppose that we have four users $S = \{4, 5, 6, 7\}$ in a SD scheme. Figure 3.8 shows the subset difference tree of the system, and Figure 3.9 shows the hierarchical tree derived from the subset difference tree as a result of algorithm 2.

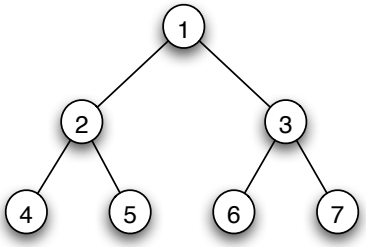

Figure 3.8: Binary tree T of a SD scheme with four users

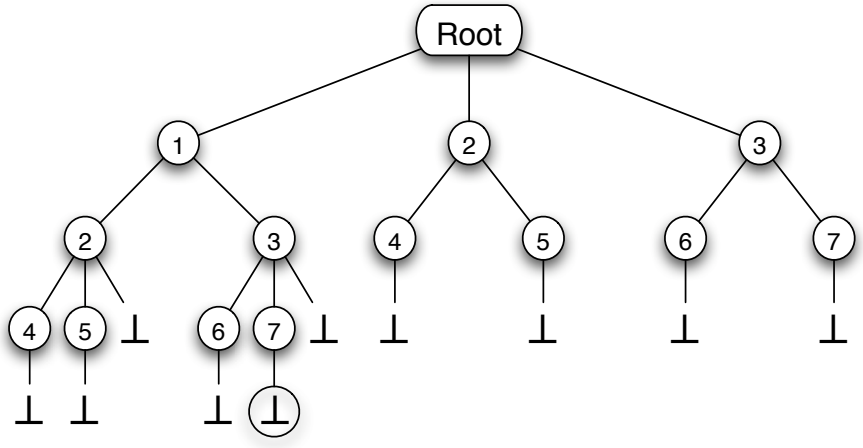

Figure 3.9: Hierarchical tree T' corresponding to T in figure 3.8

The leaf node marked with a circle in Figure 3.9 represents subset difference $\gamma(1, 7)$. Together, all of the leaf nodes represent the entire set S of the system.

The hierarchical tree can be used as a hierarchy in the previously studied BBG scheme. We first initialize the system using RootSetup to generate params and

master-key. Root then acts as the root PKG to generate private keys for all the nodes at level 1 of the hierarchical tree using the Extract algorithm. In our example, these nodes are (Root, 1), (Root, 2), and (Root, 3), and their private keys are $d_{(\text{Root},1)}$, $d_{(\text{Root},2)}$, $d_{(\text{Root},3)}$. All the nodes at level 1 correspond to all the internal nodes in the subset difference tree. Therefore, we can use their private keys as the setup keys mentioned in Section 3.3.3 to generate all the seeds for all the users. Now, we have $d_{(\text{Root},i)} = SEED_{i,i}$, where $i \in \{\text{internal nodes of } T\}$.

The private keys for the rest of the internal nodes in T' are used as seeds. As a result, to generate seeds for users, we just recursively apply the Extract algorithm (remember that in the original SD scheme this is done by using a pseudo-random sequence generator). For example, user 4 needs to have $SEED_{1,5}$, which is $d_{(\text{Root},\,1,\,2,\,5)}$. $d_{(\text{Root},\,1,\,2,\,5)}$ can be extracted from $d_{(\text{Root},\,1,\,2)}$ which in turn can be extracted from the setup-seed $d_{(\text{Root},\,1)}$.

Recall that seeds are not the long-lived keys used to directly encrypt session keys. Seeds are the secret information from which the corresponding long-lived keys can be derived. In the original SD scheme, this is achieved by processing seeds through a pseudo-random sequence generator. In this public-key scheme, the extraction of the long-lived key corresponding to $SEED_{i,j}$ is realized by applying Extract on the seed to produce a private key for $\perp$, which is a child of (Root, i, ..., j). The private key for $\perp$ serves as the long-lived key $L_{i,j}$ corresponding to $SEED_{i,j}$. For example, to obtain the long-lived key for $SEED_{1,5}$, we input (Root, 1, 2, 5, $\perp$), params, and $d_{(\text{Root},\,1,\,2,\,5)}$ to Extract which then outputs $d_{(\text{Root},\,1,\,2,\,5,\,\perp)} = L_{1,5}$. Since Extract is a probabilistic algorithm, given the same input the output (seed/long-lived key) could be different each time when the algorithm is used. This property does not pose problems for our public-key SD scheme, because any legitimate private key to a node can decrypt messages encrypted to that node.

Summing up, in this public-key SD scheme, private keys for the level 1 nodes in T' are the setup-seeds, private keys for other lower level internal nodes are the seeds, and private keys for the leaves ($\perp$) are the long-lived keys. To find a subset-

cover for the privileged users in the system, the subset-cover algorithm (Algorithm 1) for the original SD scheme can be used on T. Upon finding the cover, the session key is encrypted with the long-lived keys using the BBG Encrypt algorithm, and the message is encrypted with the session key using AES.

The public-key SD scheme and its original symmetric-key counterpart are very similar structurally. It can be applied to AACS in the same way as the original SD scheme. Recall that seeds are used as device keys and long-lived keys are used as processing keys in AACS. However, it would be hard to make the new AACS backward compatible with the existing AACS without introducing much overhead.

Analysis of public-key SD scheme

The public-key SD scheme enables everyone to broadcast information, because all that are required to make a broadcast are the system global parameters and the IDs corresponding to the subset differences covering the recipients, which are all publicly known. It might be the case that the AACS LA does not want just anyone to have the ability to make a broadcast. In this situation, the content revocation mechanism studied in Section 3.4.2 could be used to indirectly regulate broadcasters, however, this mechanism might not be very effective as it only works on an individual-title basis.

The public-key SD scheme could eliminate the risk of the server being a potential single point of failure, although it is sometimes hard to achieve if the system is large. In the original SD scheme, the server has to store all the secret information of the system. If the server is breached, the whole system could be compromised. On the other hand, the server in the public-key SD scheme does not need to store any secret information after the system is entirely initialized. For a small system, after all the users get their seeds, the server erases the master-key and it no longer stores any secret information. However, for a large system such as AACS, the erasure of the master-key might not happen until enough devices are manufactured. Since such systems are designed to have a huge number of user devices, it will generally

take a long time to reach the capacity of the system.

In addition to the above-mentioned benefits, this public-key SD scheme can be augmented to provide cryptographic features that would be hard to realize on the symmetric-key one. For example, it can be augmented to provide perfect forward secrecy using the technique proposed by Boneh and Franklin [34] or Yao et al. [120]. Although this technique is not suitable for AACS due to the fact that the user would be restricted to play a movie within a certain time frame, it could be valuable to other applications.

To determine the efficiency of our public-key SD scheme, we analyze the complexity of the message expansion and the storage requirements for both the server and the client.

(1) **Complexity of the message expansion:**

The public-key SD scheme uses the same subset-cover finding algorithm as in the original SD scheme, and the ciphertext of the BBG Encrypt algorithm has constant size regardless of the depth of the ID. Hence, the message expansion has the same complexity $4r - 2$ as in the original SD scheme.

(2) **Storage complexity at the server end:**

The server needs to store the master-key as long as there are client devices that have not obtained their secret information. During this phase, the storage requirement is $O(1)$. After all the clients are initialized, the server erases the master-key and thus does not need to store any secret information. The server does not need to store any public key information.

(3) **Storage complexity at the client end:**

The public-key SD scheme essentially uses the same key management method that is employed in the original SD scheme (under computational key assignment), and they only differ in key generation methods. Therefore, each device still needs to store $O(\log^2 n)$ seeds. In the original SD scheme, the size of the

seed is fixed according to the size of the block cipher that is used. However, the size of the seed in the public-key SD scheme is not fixed; it grows logarithmically with respect to the size of the user population. Combining the number of seeds and the size of the seed, the user device needs to store $O(\log^3 n)$ secret information. Since the client is almost always offline, it needs to store the global public key params. The size of params grows linearly with respect to the height of the hierarchical tree which is $\log n + 1$, so the size of params is $O(\log n)$. In total, the storage complexity of the client device is $O(\log^3 n + \log n) = O(\log^3 n)$.

Chapter 4

Drive-Host Authentication with Key Agreement

4.1 Introduction

To design a media protection scheme that is able to run on open platforms like PCs, designers have to make sure that the scheme is not susceptible to the "virtual device attack". A virtual device can mimic a physical hardware device in all aspects so that the CPU is tricked into believing that a device exists when actually it does not. To deploy a virtual device attack on a media system such as the DVD playback system, the attacker can build software that implements a virtual DVD drive. The content of the optical disc is moved onto the computer's hard drive as a disc image. The attacker can then play back this "DVD disc" through the virtual DVD drive on a legitimate DVD player software.

The attacker can certainly duplicate the disc image into multiple copies and disseminate them illegally, even though he/she never learns the content of the DVD in the clear. In order to defend against this attack, the drive has to have the ability to prove to the host (e.g., the playback software) that it is a legitimate drive. This can be done through a cryptographic authentication protocol.

AACS drive-host authentication scheme achieves mutual authentication, which means that not only the drive proves to the host its legitimate identity but also the host has to prove its identity to the drive. After the drive and the host complete a successful session of the authentication protocol, a shared secret key is established between them. Therefore, AACS drive-host mutual authentication protocol is combined with a key agreement protocol. The shared secret key is then used for message authentication purposes (see Section 4.2.2).

4.1.1 Mutual Authentication Protocol and Key Agreement Protocol

In a mutual authentication protocol, the two participating entities need to prove their identities to each other. If an entity has successfully proven its identity to the other entity, the other entity is required to "accept". A session of a mutual authentication protocol is a successfully completed session if both participants have accepted by the end of the session. Mutual authentication protocols can be devised by using either symmetric or asymmetric key cryptographic primitives. Stinson [101, Chapter 9] provides some good studies on mutual authentication protocols.

After two entities have authenticated themselves to each other, most likely they will want to communicate with each other. It therefore makes sense to combine a key agreement protocol or a key exchange protocol with a mutual authentication protocol, because a shared secret key provides confidentiality and/or data integrity to both communicating entities. In a key agreement protocol, both entities contribute information that is used to derive a shared secret key. A key agreement protocol most often uses asymmetric-key primitives. In a key exchange protocol, the shared secret key is not necessarily a product of the information contributed by both participating entities, and it is hard to achieve perfect forward secrecy in a key exchange protocol. We concentrate our study on key agreement protocols.

A key agreement protocol is said to provide *implicit key authentication* to both entity A and entity B if A is assured that no one other than B can possibly learn the

value of the shared secret key (likewise, B is assured that no one other than A can learn the value of the key). Note that this property does not necessarily mean that A is assured of B actually possessing the key nor is A assured that B can actually compute the key. A key agreement protocol with implicit key authentication is called an *authenticated key agreement* (AK) protocol.

A key agreement protocol is said to provide *implicit key confirmation* if A is assured that B can compute the secret key while no others can. A protocol provides *explicit key confirmation* if A is assured that B has computed the secret key and no one other than B can compute the key. A key agreement protocol that provides key confirmation (either implicit or explicit) to both participating entities is called an *authenticated key agreement with key confirmation* (AKC) protocol. For example, explicit key confirmation can be achieved by using the newly derived key to encrypt a known value and to send it to the other entity. In most cases, using a key agreement protocol with implicit key confirmation is sufficient. For more information on key agreement protocols, please refer to [101, Chapter 11].

In this chapter, we introduce the AACS drive-host authentication scheme. By performing an analysis of this scheme, we identify several weaknesses in it. Modifications of the scheme are suggested in order to provide better security. At the end, we present a semi-formal proof of security of the modified scheme.

4.2 Description of AACS Drive-Host Authentication Scheme

4.2.1 The Basic Procedure

When using AACS in a PC-based system where the drive and the host are separate entities, both the drive and the host are issued certificates from the AACS LA. These certificates, called the *drive certificate* and *host certificate*, each contain fields stating the capabilities of the device, a unique identifier, the device's public key, and

a signature from the AACS LA verifying the integrity of the certificate signed with an AACS LA private key. Both the drive and the host have the corresponding AACS LA public key for signature verification. A full description of the certificate format can be found in the AACS Introduction and Common Cryptographic Elements specification [8, Chapter 4].

Authentication between the drive and the host occurs each time new media is placed into the drive. This is necessary because the new disc may contain updated revocation lists. Each compliant disc contains a data structure called the *media key block* (MKB), which holds the necessary information needed to derive the keys to decrypt the content. It also contains the latest *drive revocation list* (DRL) and *host revocation list* (HRL) which, respectively, contain a list of IDs of the revoked drives and a list of IDs of the revoked hosts. A drive may only communicate with a host that has not been revoked, and a host may only communicate with a drive that has not been revoked.

The following is a flow representation of the AACS drive-host authentication scheme. A detailed description can be found in the specification [8, Section 4.3]. The numbers on each line correspond to the number given to that step in the specification [8, Section 4.3].

The AGID sent in step 5 is the Authentication Grant Identifier, which is used to identify a specific session in the case where a drive can support connections from multiple hosts. H_{cert} is the host certificate, and D_{cert} is the drive certificate.

	Drive		Host
1.		$\xrightarrow{MKB}$	Compare version of stored DRL to DRL in MKB. If DRL in MKB is not newer, use stored DRL.

2.			If DRL in MKB is newer, verify DRL signature. Use DRL in MKB. Abort if signature is not valid.
3.	Compare version of stored HRL to HRL in MKB. If HRL in MKB is not newer, use stored HRL.		
4.	If HRL in MKB is newer, verify HRL signature. Use HRL in MKB. Abort if signature is not valid.		
5.		$\xrightarrow{AGID}$	
6.			Generate 160-bit nonce H_n.
7.		$\xleftarrow{H_n,\ Hcert}$	
8.	Verify host certificate type and length. Abort on failure.		
9.	Verify signature on host certificate. Abort on failure.		
10.	Check HRL and abort if Host ID is found.		

11.			Request nonce D_n and drive certificate.
12.	Generate 160-bit nonce D_n.		
13.		$\xrightarrow{D_n,\ Dcert}$	
14.			Verify drive certificate type and length. Abort on failure.
15.			Verify signature on drive certificate. Abort on failure.
16.			Check DRL and abort if Drive ID is found.
17.			Request a point D_v on the elliptic curve and its signature.
18.	Generate 160-bit random value D_k.		
19.	Calculate $D_v = D_k G$ where G is the base point of the elliptic curve.		
20.	Calculate D_{sig} as the signature of $H_n \| D_v$ using the drive's private key.		

21.		$\xrightarrow{D_v,\ D_{sig}}$	
22.			Verify D_{sig} and abort on failure.
23.			Generate 160-bit random number H_k.
24.			Calculate $H_v = H_k G$.
25.			Calculate H_{sig} as the signature of $D_n \| H_v$ using the host's private key.
26.		$\xleftarrow{H_v,\ H_{sig}}$	
27.	Verify H_{sig} and abort on failure.		
28.	Calculate Bus Key B_k as the 128 least significant bits of x-coord($D_k H_v$).		
29.			Calculate Bus Key B_k as the 128 least significant bits of x-coord($H_k D_v$).

After successfully completing the drive-host authentication algorithm, the drive and the host have established a shared *bus key* based on an elliptic curve Diffie-Hellman key agreement protocol [88]. It is interesting to note that while this key could be used to encrypt messages between the drive and the host, it is not actually

used for this purpose. Instead, the bus key is used solely for message authentication by including a MAC for any message traveling between the drive and the host. The current AACS specifications require neither the drive nor the host to be capable of encrypting and decrypting bus messages, however there is a flag in each certificate stating whether or not an entity is capable of performing bus encryption.

4.2.2 Requesting Media-Specific Information

In the previous section, we mentioned that the drive and the host can establish a shared bus key. This bus key is used to ensure that communications between the drive and the host remain unaltered. In this section, we briefly describe how the bus key is used when requesting certain values from the disc, such as the *volume ID* or the *binding nonce*. These are necessary when decrypting a disc, as they are used to derive the *title key*, which in turn is used to decrypt the content.

The protocol is simply a direct exchange of messages using MACs to ensure that the message is authentic. Figure 4.1 demonstrates the process in a protocol diagram. When the host makes a request for M, the drive reads it from the disc, computes a MAC using the bus key, and sends M in plaintext along with the MAC. Upon receiving the M, the host verifies the MAC and decides whether or not the message is valid.

4.3 Analysis of AACS Drive-Host Authentication Scheme

In this section, we analyze the AACS drive-host authentication scheme. We observe several weaknesses that could lead to various attacks, and we provide corresponding improvements to strengthen the original scheme.

Our discussion of security is based on the standard security model for authentication and key agreement schemes, which was first proposed by Bellare and Ro-

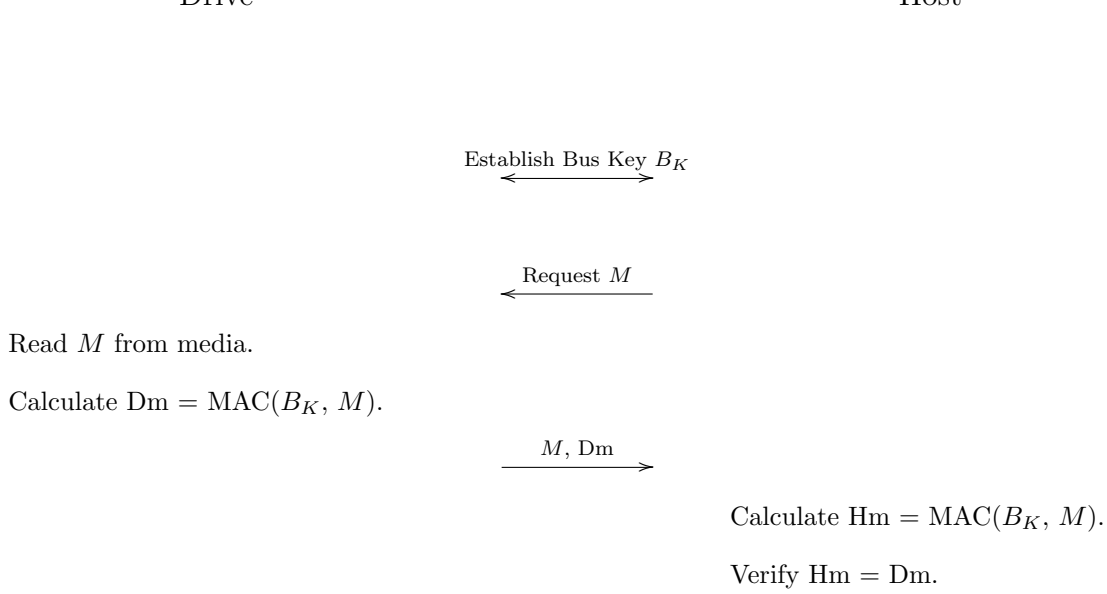

Figure 4.1: Protocol for transferring media-specific information

gaway in the symmetric-key setting [23]. Blake-Wilson et al. later generalized this model into the public-key setting [30]. In the standard model, the adversary has enormous power and controls all communication between entities. The adversary can read, modify, create, delay and replay messages, and he/she can initiate new sessions at any time.

4.3.1 Weakness 1: Design Error

This weakness is found in the first four steps of the drive-host authentication scheme. Suppose that the DRL in the MKB is newer than the DRL stored in the host. A malicious party, Oscar, can change the MKB version number to an older one, and send the modified MKB$'$ to the host. This modification might not be detected during the authentication procedure, because according to the specification, the host first checks the MKB version number, and if the version number is older than its DRL's, it skips over step 2, which involves verifying the signature on the DRL in the MKB.

If the drive has already been revoked, it could maliciously alter the MKB version number in order not to let the host update its DRL, so that it can keep interacting with the host.

The altered MKB might eventually be detected when the host processes the MKB during content decryption. However, it is undesirable for a revoked drive to be able to talk to the host until then.

The fix to this weakness is simple: The host should verify the MKB and DRL signatures before checking the version numbers. The same modification can be made to the drive side. Figure 4.2 shows the modification.

4.3.2 Weakness 2: Unknown Key-Share Attack

Suppose A and B are two honest participating entities trying to set up a shared secret key through a key agreement protocol, and O is an active malicious entity. An unknown key-share attack on a key agreement protocol is an attack through which O causes one of the two honest entities, say A, to believe that it shares a key with O, but it actually shares the key with the other honest entity B, and B believes that the key is shared with A. So, at the end of the protocol, O can act on behalf of B to interact with A. There are a number of papers studying the unknown key-share attack and its application on a number of protocols, e.g. [17], [31], [73], [97], and [121].

We can simplify the original flow representation of the drive-host authentication scheme into the one shown in Figure 4.3 by taking into consideration only the core steps involved in authentication and key agreement. A similar flow diagram is also provided in the specification [8, Section 4.3].

1. Host initiates a session with Drive. It sends a random nonce H_n and its certificate H_{cert} to Drive. Drive verifies the signature of the Host certificate

Drive		Host
1.	$\xrightarrow{MKB}$	Verify MKB and DRL signatures. Abort if signatures are not valid.
2.		Compare version of stored DRL to DRL in MKB. If DRL in MKB is not newer, use stored DRL. Otherwise, use DRL in MKB, and store it for later reference.
3. Verify MKB and HRL signatures. Abort if signatures are not valid.		
4. Compare version of stored HRL to HRL in MKB. If HRL in MKB is not newer, use stored HRL. Otherwise, use HRL in MKB, and store it for later reference.		

Figure 4.2: Improved first four steps

using the AACS LA public key. If the verification fails, Drive shall abort this authentication procedure.

2. Drive replies to the Host with a random nonce D_n and its certificate D_{cert}. Host verifies the signature of the Drive certificate using the AACS LA public key. If the verification fails, Host shall abort this authentication procedure.

3. Drive generates a 160-bit random number D_k and uses it to calculate a point

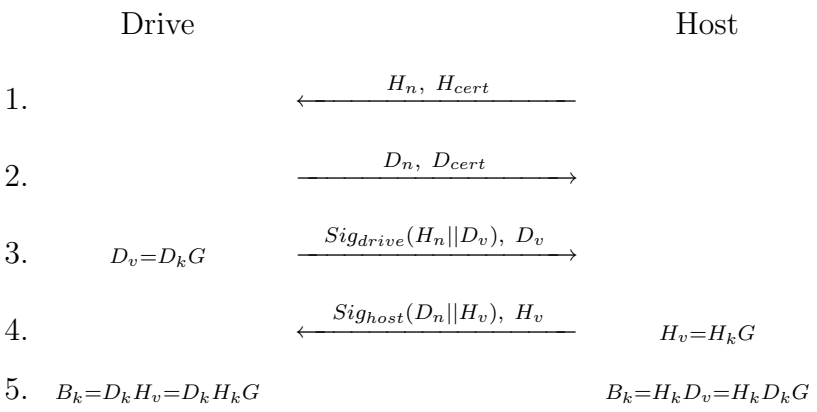

Figure 4.3: Simplified AACS drive-host authentication protocol

D_v on the elliptic curve. (G is the base point of the elliptic curve.) Drive then creates a signature of the concatenation of the nonce H_n and D_v. Drive sends the digital signature and D_v to Host. Host verifies the signature, and aborts the session on failure.

4. Host generates a 160-bit random number H_k and uses it to calculate a point H_v on the elliptic curve. Host then creates a signature of the concatenation of the nonce D_n and H_v. Host sends the digital signature and H_v to Drive. Drive verifies the signature, and aborts the session on failure.

In the last step, both Drive and Host calculate the shared secret bus key B_k.

An attacker, $\text{Drive}_{\text{Oscar}}$, which is also a legitimate drive, can use a parallel session to deploy an unknown key-share attack. Figure 4.4 shows the diagram of the attack.

The attack works in this way:

1. Host initiates a session with $\text{Drive}_{\text{Oscar}}$. It sends its random nonce H_n and certificate H_{cert} to $\text{Drive}_{\text{Oscar}}$.

2. $\text{Drive}_{\text{Oscar}}$ relays the traffic to Drive as if Host is initiating a session with Drive. Drive receives H_n and H_{cert} and verifies that H_{cert} is valid.

3. Drive sends back its random nonce D_n and certificate D_{cert} to Host, which of course get intercepted by $\text{Drive}_{\text{Oscar}}$.

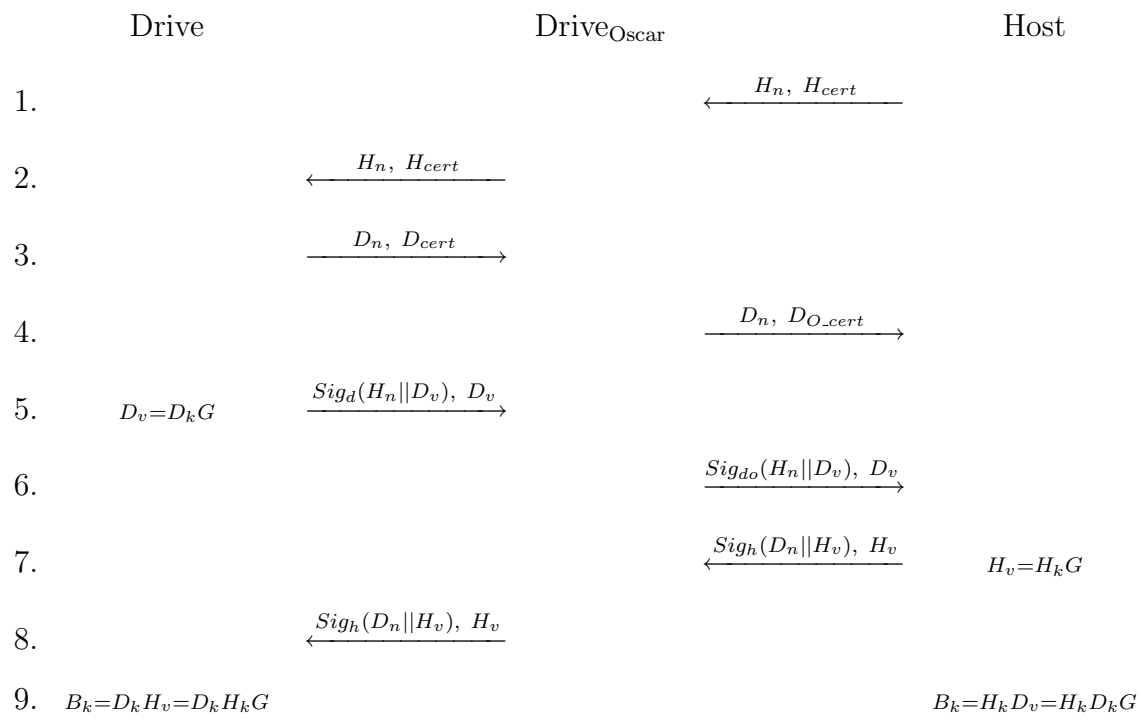

Figure 4.4: Unknow key-share attack on AACS drive-host authentication protocol

4. Drive$_{\text{Oscar}}$ relays the random nonce D_n to Host, however, it does not relay the Drive's certificate. Instead, it sends its own certificate D_{O_cert} to Host. Host receives D_{O_cert} as well as D_n. It is tricked into believing that Drive$_{\text{Oscar}}$ has generated this random nonce. Host verifies Drive$_{\text{Oscar}}$'s certificate, and the verification should pass because Drive$_{\text{Oscar}}$ is a legitimate drive.

5. Following the AACS drive-host authentication protocol, Drive generates a random number D_k and calculates a point D_v on the elliptic curve. Drive then creates a signature of the concatenation of the nonce H_n and D_v. Drive sends the digital signature and D_v to Host.

6. Drive$_{\text{Oscar}}$ relays D_v to Host. However, it creates its own signature of the concatenation of the nonce H_n and D_v using its private key. It can do so because both H_n and D_v are available to it. It sends this signature instead of Drive's signature to Host. Host verifies the signature using Drive$_{\text{Oscar}}$'s public key obtained from D_{O_cert}. The verification should pass.

7. Host generates a random number H_k and calculates a point H_v on the elliptic curve. Drive then creates a signature of the concatenation of the nonce D_n and H_v. Drive sends the digital signature and H_v to $\text{Drive}_{\text{Oscar}}$.

8. $\text{Drive}_{\text{Oscar}}$ relays the traffic to Drive. Drive verifies the signature, and the verification should pass.

By the time the session is complete, Drive has accepted Host, and it can calculate the shared bus key B_k. On the other hand, Host does not accept Drive because it simply does not know the existence of Drive from this interaction. Instead, it has accepted $\text{Drive}_{\text{Oscar}}$. Host can also calculate the same shared bus key B_k.

Although $\text{Drive}_{\text{Oscar}}$ does not know the secret bus key B_k in the end, it has tricked Host into believing that it shares the bus key with $\text{Drive}_{\text{Oscar}}$. Host thinks that it is talking to $\text{Drive}_{\text{Oscar}}$ while actually it is interacting with Drive.

This attack could practically be exploited. For example, suppose that Drive_{A} is revoked. Then it can employ this attack to ask Drive_{B}, which is not revoked, to impersonate it. Since the host only sees Drive_{B}'s certificate, the authentication procedure should complete successfully. In this way, Drive_{A} can still interact with the host after the authentication procedure. It has effectively bypassed the authentication procedure.

Such an attack is enabled due to the fact that in the last two flows $\text{Drive}_{\text{Oscar}}$ can simply copy the traffic. This problem can be fixed by including the entity IDs in the signature. (See Section 4.3.4).

4.3.3 Weakness 3: Man-In-The-Middle Attack

The adversarial goal in an attack to a mutual authentication protocol is to cause an honest participant to "accept" after a flow in which the adversary is active. To consider a mutual authentication protocol secure, it has to satisfy the following two conditions:

1. Suppose A and B are the two participants in a session of the protocol and they are both honest. Suppose also that the adversary is passive. Then A and B will both "accept".

2. If the adversary is active during a given flow of the protocol, then no honest participant will "accept" after that flow.

Figure 4.5 shows an attack which might not be as powerful and practical as the previous one. Nonetheless, it shows a weakness in this protocol.

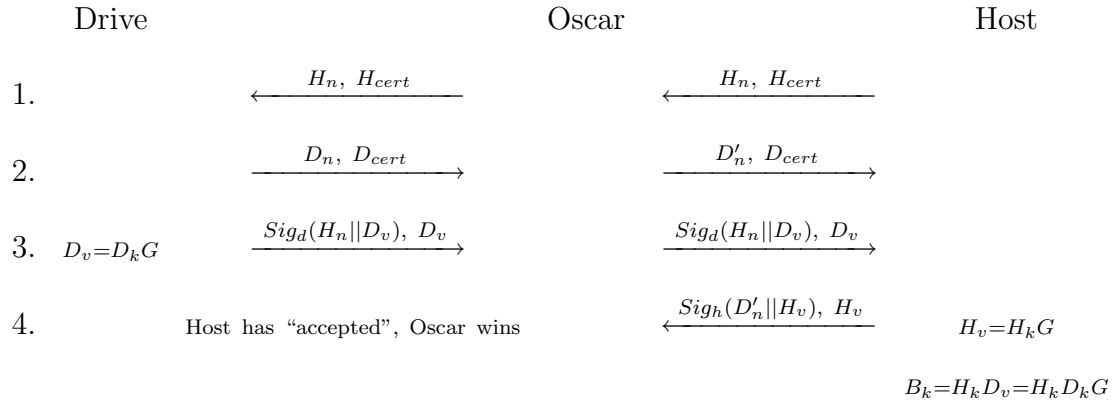

Figure 4.5: A trivial Man-In-The-Middle attack

In this case, Oscar could be a polynomial time adversary with the ability to listen and to modify the traffic. Notice that in step 2 when Oscar relays the traffic from Drive to Host, it modifies the random nonce D_n generated by Drive into a different one D'_n. This does not make Host terminate the session. In step 3, after Host has successfully verified Drive's signature, it "accepts". This violates condition 2 mentioned above, hence the protocol should not be considered secure.

4.3.4 Improved Scheme

Since the scheme makes use of certificates, we can improve it using a simplified *Station-to-Station* key agreement protocol (STS). STS protocol is a key agreement scheme based on Diffie-Hellman scheme that provides mutual authentication. For more information on STS protocols, please refer to [53], [101, Chapter 11].

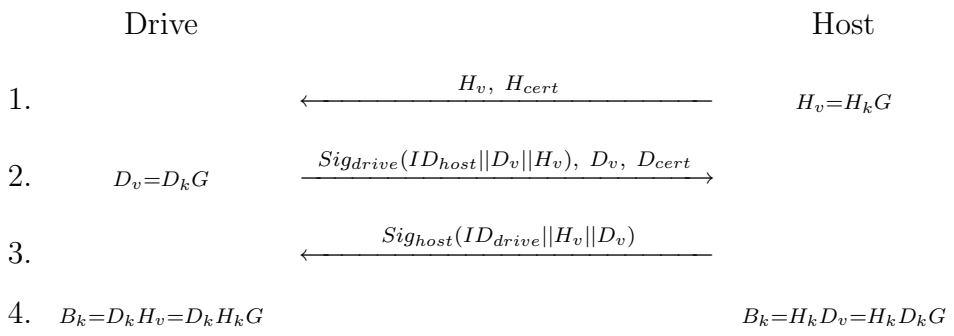

Figure 4.6: Improved scheme based on the Station-to-Station protocol

Figure 4.6 shows the modified drive-host authentication scheme based on STS. This modification solves both problems stated in weakness 2 and 3 (a security proof is given in the next section). In addition, it improves the efficiency of the original protocol, because the number of interactions between Drive and Host is reduced.

1. Host initiates a session with Drive. It generates a 160-bit random number H_k and uses it to calculate a point H_v on the elliptic curve. It sends the H_v and its certificate H_{cert} to Drive. Drive verifies the signature of the Host certificate using the AACS LA public key. If the verification fails, Drive shall abort this session.

2. Drive generates a 160-bit random number D_k and uses it to calculate a point D_v on the elliptic curve. Drive then creates a signature of the concatenation of the Host ID, D_v, and H_v. Drive sends the digital signature, D_v, and its certificate D_{cert} to Host. Host verifies the signature created by Drive: $ver_{drive}(ID_{host}||D_v||H_v, \text{Drive}'\text{s signature}) \stackrel{?}{=} \text{true}$, and it also verifies the signature of the Drive certificate. If either of the two verifications fail, Host shall abort the session.

3. Host creates a signature of the concatenation of the Drive ID, H_v, and D_v and sends it to Drive. Drive verifies the signature: $ver_{host}(ID_{drive}||H_v||D_v, \text{Host}'\text{s signature}) \stackrel{?}{=} \text{true}$, and aborts the session on failure.

At the end of the protocol, both Drive and Host are able to establish the shared

secret bus key B_k. Points H_v and D_v in this protocol also play a role as random challenges.

The following flow representation shows the entire modified drive-host authentication protocol.

	Drive		Host
1.		$\xrightarrow{MKB}$	Verify MKB and DRL signatures. Abort if signatures are not valid.
2.			Compare version of stored DRL to DRL in MKB. If DRL in MKB is not newer, use stored DRL. Otherwise, use DRL in MKB, and store it for later reference.
3.	Verify MKB and HRL signatures. Abort if signatures are not valid.		
4.	Compare version of stored HRL to HRL in MKB. If HRL in MKB is not newer, use stored HRL. Otherwise, use HRL in MKB, and store it for later reference.		
5.		$\xrightarrow{AGID}$	

6.			Generate 160-bit random number H_k.
7.			Calculate $H_v = H_k G$ where G is the base point of the elliptic curve.
8.		$\xleftarrow{H_v,\ Hcert}$	
9.	Verify host certificate type and length. Abort on failure.		
10.	Verify signature on host certificate. Abort on failure.		
11.	Check HRL and abort if Host ID is found.		
12.			Request a point D_v on the elliptic curve, signature, and drive certificate.
13.	Generate 160-bit random value D_k.		
14.	Calculate $D_v = D_k G$ where G is the base point of the elliptic curve.		

15.	Calculate D_{sig} as the signature of $ID_{host} \| D_v \| H_v$ using the drive's private key.		
16.		$\xrightarrow{D_{sig},\ D_v,\ Dcert}$	
17.			Verify drive certificate type and length. Abort on failure.
18.			Verify signature on drive certificate. Abort on failure.
19.			Check DRL and abort if Drive ID is found.
20.			Verify D_{sig} and abort on failure.
21.			Calculate H_{sig} as the signature of $ID_{drive} \| H_v \| D_v$ using the host's private key.
22.		$\xleftarrow{H_{sig}}$	
23.	Verify H_{sig} and abort on failure.		
24.	Calculate Bus Key B_k as the 128 least significant bits of x-coord($D_k H_v$).		

25. Calculate Bus Key B_k as the 128 least significant bits of x-coord($H_k D_v$).

The new protocol solves all the aforementioned problems. Since the random challenges H_n and D_n are omitted, it enables the drive and the host to perform fewer interactions, and is therefore more efficient.

4.4 Security of the Modified Drive-Host Authentication Scheme

The modified scheme protects against the unknown key-shared attack mentioned earlier.

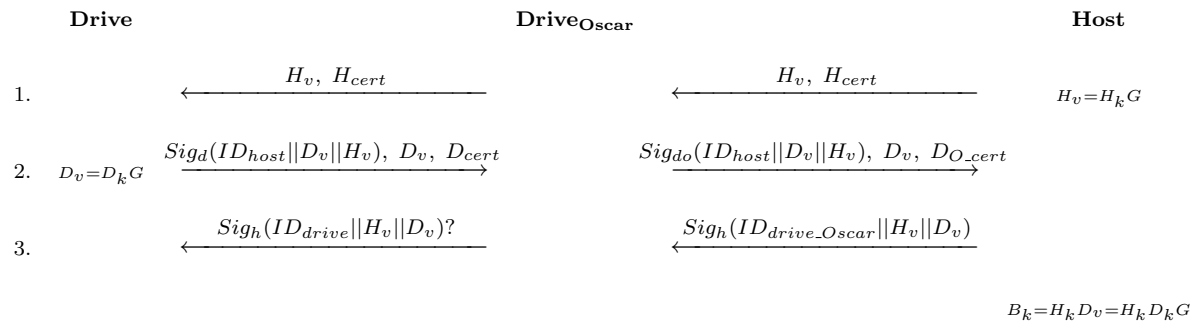

Figure 4.7: Protection against unknown key-share attack

In Figure 4.7, a question mark following a signature indicates that the adversary is unable to compute this signature. At step 3, the signature which Host sends to $\text{Drive}_{\text{Oscar}}$ contains $\text{Drive}_{\text{Oscar}}$'s ID not Drive's ID because Host believes that it is talking to $\text{Drive}_{\text{Oscar}}$. $\text{Drive}_{\text{Oscar}}$ cannot compute Host's signature on the string $ID_{drive}||H_v||D_v$ because he does not know Host's private signing key. As a result, unknown key-share attack is thwarted.

After step 2, Host "accepts" the authentication because it should successfully verify $\text{Drive}_{\text{Oscar}}$'s signature and certificate. This does not violate the second condition of considering a mutual authentication protocol secure mentioned in Section 4.3.3, because Host is authenticating with $\text{Drive}_{\text{Oscar}}$.

The modified scheme also protects against man-in-the-middle attack.

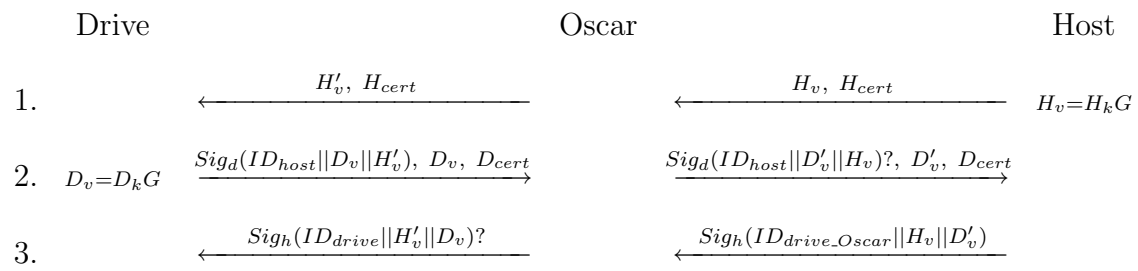

Figure 4.8: Protection against man-in-the-middle attack

As shown in Figure 4.8, if Oscar modifies H_v, he then would not be able to produce Host's signature on $ID_{drive}||H'_v||D_v$ because he does not know Host's private signing key. Likewise, if Oscar modifies D_v, he then would not be able to produce Drive's signature on $ID_{host}||D'_v||H_v$ because he does not know Drive's private signing key.

Of course, we want to show that the modified scheme is secure against all possible attacks, not just two particular attacks. Hence, we need to show that the modified scheme is a secure mutual authentication scheme, and that it provides assurances regarding knowledge of the shared secret key. For the proof of security of our improved scheme, an informal treatment based on [101, Chapter 11] is given in the rest of this section.

4.4.1 Secure Mutual Authentication

A secure mutual authentication has to satisfy the two conditions described in Section 4.3.3. Let us first show that our modified scheme satisfies the first condition.

Since no one is modifying the traffic, if the adversary is passive and the two participants are honest, they should successfully authenticate themselves to each

other and both compute the shared secret key as in the Diffie-Hellman key agreement scheme. Assuming the intractability of the *Decision Diffie-Hellman* problem, the inactive adversary cannot compute the share secret key.

To prove that our modified scheme satisfies the second condition, let us assume that the adversary is active. The adversary wants to fool at least one of the two participants to "accept" after a flow in which he/she is active. We show that the adversary will not scceed in this way, except with a very small probability.

Definition 4.4.1. A signature scheme is (ϵ, Q, T)-secure if the adversary cannot construct a valid signature for any new message with probability greater than ϵ, given that he/her has previously seen at most Q different valid signatures, and given that his/her computation time is limited to T.

Definition 4.4.2. A mutual authentication scheme is (ϵ, Q, T)-secure if the adversary cannot fool any honest participants into accepting with probability greater than ϵ, given that he/she has observed at most Q previous sessions between the honest participants, and given that the his/her computation time is at most T.

Time T is usually chosen to be very long so that by the time the adversary successfully computes the correct result the value of the result has decreased to an insignificant level. For simplicity of notation, we omit the time parameter. Q is a specified security parameter. Depending on the application, it could be assigned with various values. The probability ϵ is usually chosen to be so small that the chance of success is negligible.

Theorem 4.4.1. *Suppose that Sig is an (ϵ, Q)-secure signature scheme, and suppose that random challenges H_v and D_v are k bits in length. Then the scheme shown in Figure 4.6 is a $(Q/2^{k-1} + 2\epsilon, Q)$-secure mutual authentication scheme.*

Proof. The adversary, Oscar, observes Q previous sessions of the protocol before making his attack. A successful attack by Oscar is to deceive at least one honest participant in a new session into accepting after he is active in one or more flows.

1. Oscar tries to deceive Host. In order to make Host accept, it has to receive a signature signed by Drive containing the Host ID and the random challenge H_v. There are only two ways for Oscar to acquire such a signature: either from a previously observed session or by computing it himself.

   To observe such a signature from a previous session, H_v has to be used in that session. The probability that Host has already used the challenge in a specific previous session is $1/2^k$. There are at most Q previous sessions under consideration, so the probability that H_v was used as a challenge in one of these previous sessions is at most $Q/2^k$. If this happens, Oscar can re-use Drive's signature and D'_v (which may or may not be the same as D_v) from that session to fool Host.

   To compute such a signature himself, Oscar has at most a chance of ϵ, since Sig is (ϵ, Q)-secure.

   Therefore, Oscar's probability of deceiving Host is at most $Q/2^k + \epsilon$.

2. Oscar tries to deceive Drive. This is quite similar to the case we have discussed above. In order to fool Drive, Oscar has to have a legitimate signature signed by Host. As in the previous case, the two ways for Oscar to acquire such a signature are either from a previously observed session or by computing it himself.

   To observe such a signature from a previous session, Oscar re-uses a H_v from a previous session S to send to Drive, and hopes that Drive will reply with the same D_v as in S so that he can re-use the corresponding signature. This happens with probability $1/2^k$. The best case scenario for the adversary would be that all Q previously observed sessions have the same H_v. Because if any D_v from the Q sessions is re-used by Drive, Oscar can then re-use the corresponding signature to fool Drive. Hence, Oscar has at most $Q/2^k$ probability to re-use Host's signature to deceive Drive.

   Again since Sig is (ϵ, Q)-secure, Oscar can compute such a signature with a probability of at most ϵ.

Therefore, Oscar's probability of deceiving Drive is at most $Q/2^k + \epsilon$.

Summing up, the probability for Oscar to deceive one of Host or Drive is at most $(Q/2^k + \epsilon) + (Q/2^k + \epsilon) = Q/2^{k-1} + 2\epsilon$. □

4.4.2 Implicit Key Confirmation

Now, let us see what we can infer about the modified scheme if Host or Drive "accepts". Firstly, suppose that Host "accepts". Because the modified scheme is a secure mutual authentication scheme, Host can be confident that it has really been communicating with Drive and that the adversary was inactive before the last flow. Assuming that Drive is honest and that it has executed the scheme according to the specifications, Host can be confident that Drive can compute the value of the secret bus key, and that no one other than Drive can compute the value of the bus key.

Let us consider in more detail why Host should believe that Drive can compute the bus key. The reason for this belief is that Host has received Drive's signature on the values H_v and D_v, so it is reasonable for Host to infer that Drive knows these two values. Now, since Drive is a honest participant and executed the scheme according to the specifications, Host can infer that Drive knows the values of D_k. Drive is able to compute the value of the bus key, provided that it knows the values of H_v and D_k. Of course, there is no guarantee to Host that Drive has actually computed the bus key at the moment when Host "accepts". We can be sure that no one else can compute the bus key because D_k is meant to be known to Drive only.

The analysis from the point of view of Drive is very similar. If Drive "accepts", then it is confident that it has really been communicating with Host, and that the bus key can be computed only by Host and no one else.

The modified scheme does not make immediate use of the new bus key, so we do not have explicit key confirmation. However, it does achieve implicit key confir-

mation. To achieve explicit key confirmation, we can use a message authentication code (MAC) on the IDs and the Diffie-Hellman parameters in the signatures, as shown in Figure 4.9. The MAC takes as its key the shared secret or a value derived from the shared secret. This protocol, which is very similar to the STS, is called SIGMA [77].

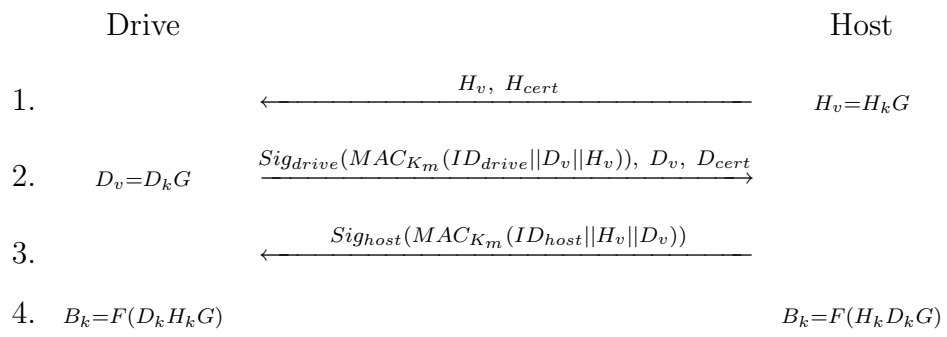

Figure 4.9: Improved scheme based on the SIGMA protocol

The key used in the MAC, K_m, is derived from the shared secret, $D_K H_K G$, through an one way function. B_k could be the shared secret, however, people have argued that it should also be derived from the shared secret through a one way function, and K_m and B_k should be "computationally independent" [77].

Chapter 5

Traitor Tracing

5.1 Introduction

In Chapter 3, we showed that broadcast encryption is the main technology in safeguarding distributed multimedia content from piracy; it encrypts the broadcast in such a way that only a set of privileged users can decrypt it. However, it is very difficult, if not impossible, to design a broadcast system to be so strong that it can guarantee no piracy of the distributed content. It is therefore prudent to assume that piracy will occur, and countermeasures that deal with such situations should be designed beforehand. Most broadcast encryption schemes today provide a means to reduce the damage caused as a result of piracy. For example, the subset revocation scheme that we have studied in Chapter 3 comes with a revocation mechanism, which gives the authority the ability to revoke compromised devices. However, this begs the following question: How can the authority know which devices are compromised in the first place? Traitor tracing technology gives an answer to this question. It allows the authority to trace piracy back to its original source, and whatever devices are involved in piracy can then be revoked. Traitors are people/devices that aid pirates in attempting to gain access to an encrypted broadcast message. In this chapter, we abstractly assume that traitors are sets of compromised device keys from compromised devices.

AACS is among the first content distribution systems to employ traitor tracing technology. The traitor tracing scheme in AACS enables the AACS LA to determine which playback device has had its device keys compromised so that they may be revoked. However, at the time of writing, this feature has yet to be commercially implemented for HD-DVD and Blu-ray discs.

There are mainly two models for pirate attacks. The first one is that the attackers compromise some legitimate devices and obtain the secret device keys. They then use the secret device keys to construct a *pirate decoder* and distribute it so that everyone can use this decoder to extract the clear content from the protected media. This model is called a *pirate decoder attack* or *key attack*. The software DeCSS which can effectively circumvent CSS and allow the plaintext content of a DVD to be extracted is essentially a pirate decoder. The authority must find the keys involved in a pirate decoder and revoke them in order to make the decoder ineffective. A common method for learning which device keys are in the decoder is to input carefully crafted ciphertext into the decoder and analyze the output. Many studies in traitor tracing deals with this threat model; e.g., [37, 44, 45, 89, 90, 39].

The second model is that the attackers compromise some legitimate devices, and instead of distributing the device keys (pirate decoder) they distribute the decrypted content obtained by using the compromised device keys. This is called a *content attack*. In this model the only forensic data is the pirated content. Some studies in traitor tracing are based on this threat model [58, 64, 93, 95, 99, 105]. To combat content attacks, "digital marks" are embedded into the content in some not easily detected manner in order to make different versions of the content for distribution. Each different version is distributed to a different user. If some version of the content is pirated and disseminated illegally, then the tracing authority can link the pirated content to its original user based on its marks. For our study, we will focus on this model, and to simplify our discussion, we will use digital movies as digital content.

The attacker can also attack the system by breaking its cryptographic primitives

in real time. With stronger cryptographic primitives employed by modern content distribution systems, this becomes harder for the attacker to do.

In the first half of this chapter, we study how multimedia content can be modified to carry forensic data, which in turn can be used to trace compromised devices. Several combinatorial traitor tracing schemes using codes with certain structures are studied, and one of them serves as the foundation for the AACS traitor tracing scheme. In the second half of this chapter, we concentrate on the study of the AACS traitor tracing scheme. In particular, we analyze the traceability of the AACS traitor tracing scheme, and we also study a few implementation issues of the scheme.

5.2 Background

Most traitor tracing schemes are designed with the help of *digital watermarking*. The tracing authority inserts digital watermarks, which are essentially some additional digital information, into the movie. The goal is to identify the device keys involved in piracy by extracting and analyzing the watermarks in a pirated movie. This kind of watermarking is known as *forensic watermarking*. A set of watermarks is uniquely correlated with a set of device keys. A device with a different set of device keys should not be able to process the content watermarked for another set of device keys. Just as fingerprints are unique for each person, we call forensic watermarks that are unique for each copy of a movie *digital fingerprints*. Digital fingerprinting was first studied by Wagner [115].

A fingerprint is mathematically modeled as a string of *symbols* from an *alphabet* Q. A symbol can be embedded in a segment of a movie to produce a variation of that segment. The number of variations of a segment is equal to the size of the alphabet. Figure 5.1 shows an example of a movie M fingerprinted into two different copies. Three segments of M are chosen to be fingerprinted. Each segment is a 10-minute snippet, and there are three different versions of each segment. Each symbol in

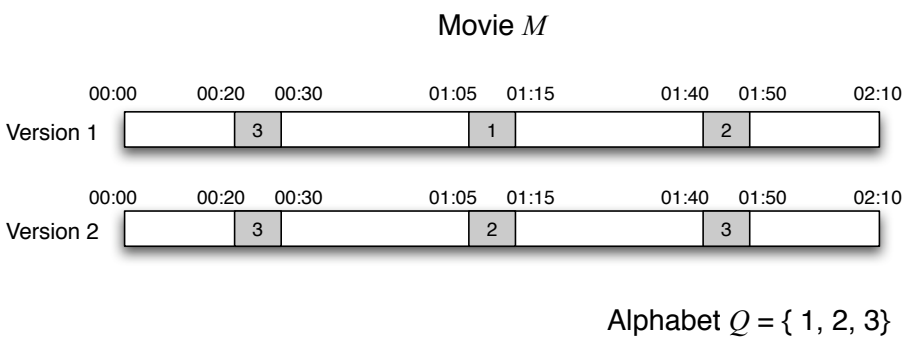

Figure 5.1: Fingerprint a movie into two different versions.

the alphabet is represented by a number. Fingerprinting a movie effectively creates different versions of the movie. Each version of the movie can be referred to by its fingerprint. In the example, version 1 of M is $(3, 1, 2)$, and version 2 is $(3, 2, 3)$. A fingerprint is uniquely associated with a set of device keys. Assume that the set of device keys in Alice's device corresponds to version 1 of M; then Alice's device can only play back version 1 of M and not any other version. On the other hand, no device other than Alice's can play back version 1 of M. If any version of M is pirated, the tracing authority can extract the fingerprint to identify which version it is and then trace back to the device that can play this version of M.

However, the trivial fingerprinting method we have just mentioned above is undesirable for physical content distribution systems. It is simply impractical to produce each individual copy of a movie with a unique digital fingerprint. It would be time consuming, and the cost would be too high for a large scale distribution such as Blu-ray. And the worst of it is that the user has to purchase the copy bearing the digital fingerprint that corresponds to the device keys in his/her device, otherwise the device would not be able to properly play back the movie. This restriction makes physical content distribution almost impossible to carry out. A common solution to this problem is that the authority puts all digital fingerprints in the system onto the digital media, and therefore the content of all the copies of the movie are the same. On the other side of the spectrum, each playback device is only able to access the content on the disc that is fingerprinted to its device

keys. If a set of device keys is compromised and is used in pirating a movie, the pirated movie would only carry the digital fingerprint corresponding to the set of compromised device keys. Figure 5.2 shows an example which illustrates this idea in more detail. Each segment of M is fingerprinted with all the symbols in the

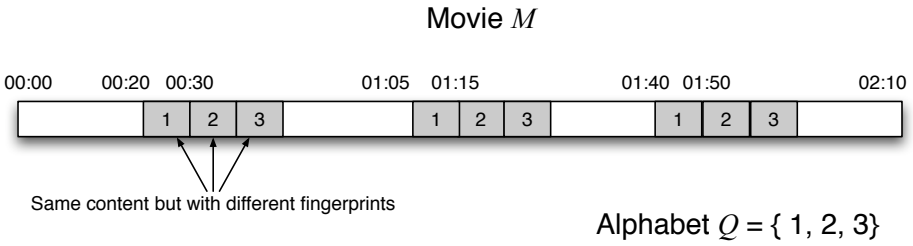

Figure 5.2: Fingerprinted movie with all versions included

alphabet, and then each variation of a segment is encrypted with a different key. All the encrypted variations are included on the disc for distribution. If Alice's device corresponds to version $(3, 1, 2)$, then her device is only able to decrypt the first segment fingerprinted with symbol 3, the second segment fingerprinted with symbol 1, and so on. There is an overhead introduced by this approach, because the content of a segment needs to be duplicated into many copies (variations). The length of the segment and the cardinality of the alphabet determine the size of the overhead.

Ideally, we would like to embed digital fingerprints into the digital content in a way that is invisible to everyone except the tracing authority. This is hard to realize, even though there are some information hiding techniques that can be used to hide the fingerprints [74]. Attackers colluding together are able to learn the positions of the fingerprints [29]. For the first fingerprinting approach, in which a movie is fingerprinted into different copies, the bits that are different among the copies of the movie can be identified as fingerprint bits. For the second fingerprinting approach, in which a movie carries all the fingerprints in the system, attackers cannot detect the fingerprints by comparing the copies of a movie because they are the same. However, fingerprints can still be detected once some copies of the movie are pirated, because each pirated copy contains a unique fingerprint. In the

literature, the property that a coalition of attackers can detect a symbol only if the symbol differs among their copies is called *fidelity* [85]. For our study, we will assume that attackers know the positions of the fingerprinted content in the movie.

Given that attackers know how to locate the fingerprint in a movie, it is natural to assume that they will tamper with it. There are two different assumptions about the possible actions of attackers.

General Assumption: Given two variations v_1 and v_2 of a segment, the attacker can generate any legitimate variation out of v_1 and v_2. The attacker can also erase the symbol on any variation or generate a variation on which the symbol is not defined in the alphabet.

Restricted Assumption: Given two variations v_1 and v_2 of a segment, the attacker can use only one of the two variations.

The restricted assumption is based on *watermarking robustness* which states that the user cannot remove or modify the watermark on a segment variation. There is some work done in this direction [13, 49]. Digital watermarking is outside the scope of this work; for more information on it, refer to [50]. We adopt the restricted assumption made by AACS, which assumes that the attacker cannot alter the fingerprint information on a variation.

We can now state the attack model of the attackers. A coalition of attackers can mix-and-match the variations on their pirated movies to make a new copy of a movie that carries an illegal fingerprint that is potentially untraceable. This attack model is called a *collusion attack*. For example, two attackers have two pirated copies of a movie with fingerprints $A = (3, 1, 2)$ and $B = (3, 2, 3)$, respectively. They can collaboratively produce a copy with fingerprint $(3, 2, 2)$ by simply copying the third segment with symbol 2 from A to the same segment in B. To thwart such collusion attacks, traitor tracing schemes are developed to enable tracing to at least some members in the coalition based on the *hybrid fingerprint*.

Traitor tracing was first studied by Chor et al. [44] in 1994, and since then it has received much research attention. There are generally two categories of traitor tracing schemes: combinatorial traitor tracing schemes [25, 44, 45, 58, 59, 91, 93, 95, 99, 104, 105, 106], and algebraic traitor tracing schemes [12, 35, 37, 39, 55, 64, 78, 76, 86, 90]. Algebraic traitor tracing uses public key techniques, and is not discussed in this work. Combinatorial traitor tracing uses a digital fingerprint as the basic structure and employs combinatorial methods to allow tracing of illegally pirated movies. In this chapter, we use traitor tracing to refer to combinatorial traitor tracing. Since traitor tracing is meant to complement broadcast encryption, some work has been done to seamlessly integrate traitor tracing into broadcast encryption [39, 59, 89, 106].

There have been many studies in the area of traitor tracing. A traitor tracing scheme usually consists of two steps:

(1) Assign variations of each segment to a user with the goal of creating a link between a copy of the multimedia content and the user.

(2) Based on the pirated content, trace back to the traitors.

If the first step of a scheme can be updated based on observed pirated content, the scheme is dynamic. *Dynamic traitor tracing* was first studied by Fiat and Tassa [58]. It involves real-time computation to determine the new assignment of variations to a user. Doing dynamic tracing is impractical in physical content distribution systems with stateless playback devices. On the other hand, Safavi-Naini and Wang [93] suggested *sequential traitor tracing*, the model that AACS uses, which requires no real-time computation. The first step is pre-detemined before the content is broadcasted and does not change afterwards. Sequential traitor tracing detects traitors sequentially. Once a colluder is identified, his/her device is revoked and the tracing authority continues to trace the remaining colluders.

The result of a tracing scheme can be either deterministic or probabilistic. A *deterministic tracing scheme* detects traitors with certainty without incriminating

an innocent user. On the other hand, a *probabilistic tracing scheme* has a small chance to incriminate an innocent user. For example, the traitor tracing schemes in [44, 45] are static and probabilistic, and the schemes in [93, 99] are static and deterministic.

AACS is the first physical content distribution system that uses traitor tracing technology. The traitor tracing scheme it uses is based on a scheme proposed by Stinson and Wei [105]. This tracing scheme is static and deterministic. It allows at least one of the colluders to be identified.

5.3 Combinatorial Traitor Tracing

Let us now formalize the definition of digital fingerprints. All the fingerprints in a system can be modeled by a *code*, and each fingerprint is a *codeword* in the code.

Definition 5.3.1. Given an alphabet Q, an (l, n, q)-*code* C is a collection of n codewords with length l over the alphabet of size q: $C \subseteq Q^l$. Each codeword represents a fingerprint, and each fingerprint is uniquely assigned to a user.

A codeword has the form $x = (x_1, x_2, \ldots, x_l)$, where $x_i \in Q$, $1 \leq i \leq l$. Hybrid fingerprints can be constructed by mixing-and-matching codewords in a code. We call the set of all hybrid fingerprints that can be constructed from a set of codewords C_0 the *descendant code* of C_0, which is denoted by $\mathbf{dc}(C_0)$.

$$\mathbf{dc}(C_0) = \{x \in Q^l \mid x_i \in \{a_i \mid a \in C_0\}, 1 \leq i \leq l\}.$$

For example, let $C_0 = \{(3, 1, 2), (3, 2, 3)\}$. The descendant code of C_0 is

$$\mathbf{dc}(C_0) = \{(3, 1, 2), (3, 2, 3), (3, 2, 2), (3, 1, 3)\}.$$

We can generalize the notion of descendant code into *w-descendant code* of C where w is a positive integer and C is a code. The w-descendant code of C, denoted

by $\mathbf{dc}_w(C)$, consists of all hybrid fingerprints that could be constructed by a set of at most w codewords in C.

$$\mathbf{dc}_w(C) = \bigcup_{C_0 \subseteq C, |C_0| \leq w} \mathbf{dc}(C_0).$$

A weak traitor tracing scheme is designed to prevent a group of colluders from "framing" an innocent user. A strong traitor tracing scheme allows at least one of the colluders to be identified. All of these schemes can be created based on codes having certain combinatorial properties. In this section, we study three types of codes that provide traceability ranging from weak to strong.

5.3.1 Frameproof and Secure Frameproof Codes

One thing a coalition of attackers can do is to frame an innocent user by constructing a copy of a pirated movie with a fingerprint of some user that is not in the coalition. It is desirable for a traitor tracing scheme to be *frameproof*; otherwise, the scheme would be useless. Boneh and Shaw first introduced the concept of w-frameproof codes, which have the property that no coalition of at most w attackers can frame a user not in the coalition [38].

Definition 5.3.2. An (l, n, q)-code C is a w-frameproof (w-FP) code if $x \in \mathbf{dc}(C_0) \cap C$ implies $x \in C_0$, for every $C_0 \subseteq C$ such that $|C_0| \leq w$.

Thus, in a w-frameproof code, the only codewords in the descendant code of a coalition of at most w attackers are the codewords of the members of the coalition.

For example, $C = \{(3,1,2),(3,2,3),(1,1,3)\}$ is 2-FP, whereas $C' = \{(3,1,2),(3,2,3),(1,2,3)\}$ is not, because $(3,2,3) \in \mathbf{dc}(\{(3,1,2),(1,2,3)\})$. As suggested by Boneh and Shaw [38], a $b \times b$ identity matrix is a b-FP $(b, b, 2)$-code by viewing the rows of the matrix as codewords. This construction is not very useful because the code length grows linearly in the number of users. Boneh and Shaw also give a construction for a w-FP $(b, 2^{\sqrt{b}/w}, 2)$-code for any positive integer b and w using

a concatenation of frameproof codes with error-correcting codes. They also prove that there exists a w-FP $(b, 2^{b/(16w^2)}, 2)$-code for any positive integer b and w without giving an explicit construction. More explicit constructions of FP codes are given by Stinson and Wei [105] using combinatorial structures and perfect hash families, and new upper bounds on the size of such codes are presented by Sarkar and Stinson [95].

The frameproof codes being considered in [38, 95, 105] are over a binary alphabet. This was later generalized to q-ary alphabets by Staddon et al. [99]. In that paper, the upper bound on the size of a w-FP (l, n, q)-codes is improved to

$$n \leq w \left(q^{\lceil \frac{l}{w} \rceil} - 1 \right).$$

This bound is further improved by Blackburn [28] to

$$n \leq t q^{\lceil \frac{l}{w} \rceil} + O(q^{\lceil \frac{l}{w} \rceil - 1})$$

where $1 \leq t \leq w$ and $t = l \pmod{w}$.

There are strong connections between *separating hash families* and frameproof codes. We now show a popular construction of frameproof codes using separating hash families [104].

Definition 5.3.3. Let n and q be positive integers and $n \geq q$. An (n, q)-hash family is a set of functions, $\mathcal{H}$, such that $|X| = n$, $|Y| = q$, $h : X \to Y$ for each $h \in \mathcal{H}$. We use the notation $\mathrm{HF}(l; n, q)$ to denote an (n, q)-hash family with $|\mathcal{H}| = l$.

An $\mathrm{HF}(l; n, q)$ can be depicted as an $n \times l$ table with entries from the set Y, where each column of the table corresponds to one of the functions in $\mathcal{H}$. Similarly, we can represent an (l, n, q)-code, C, as an $n \times l$ table over an alphabet with q symbols, where each row of the table is a codeword in C. The table representations of the code and the hash family reveal an interesting connection between the two. Given an (l, n, q)-code C with $n \geq q$, we define $\mathcal{H}(C)$ to be the $\mathrm{HF}(l; n, q)$ whose table representation is the same as the table representation of C. The rows of the table can be interpreted as codewords in C, and the columns of the table can be viewed as functions in $\mathcal{H}$.

Definition 5.3.4. Let n, q, w_1 and w_2 be positive integers and let $n \geq q$. An $(n, q, \{w_1, w_2\})$-separating hash family is an (n, q)-hash family, $\mathcal{H}$, such that for any $X_1, X_2 \subseteq X$ with $|X_1| = w_1$, $|X_2| = w_2$ and $X_1 \cap X_2 = \emptyset$, there exists at least one $h \in \mathcal{H}$ such that

$$\{h(x) \mid x \in X_1\} \cap \{h(x) \mid x \in X_2\} = \emptyset.$$

We use the notation $\mathrm{SHF}(l; n, q, \{w_1, w_2\})$ to denote an $(n, q, \{w_1, w_2\})$-separating hash family with $|\mathcal{H}| = l$.

An $\mathrm{SHF}(l; n, q, \{w_1, w_2\})$ is a hash family and hence it can be represented by an $n \times l$ table. The special property of this table is that in any two disjoint sets of rows A and B with $|A| = w_1$ and $|B| = w_2$, there exists at least one column such that the entries in A are distinct from the entries in B.

The connection between frameproof codes and separating hash families is stated in the following theorem. It shows a way of constructing frameproof codes from separating hash families.

Theorem 5.3.1. *[104] An (l, n, q)-code, C, is a w-frameproof code if and only if $\mathcal{H}(C)$ is an* $\mathrm{SHF}(l; n, q, \{w, 1\})$.

Proof. Let C be represented by an $n \times l$ table T over an alphabet of q symbols, where each row is a codeword in C. Following from the definition, C is w-frameproof if and only if the symbol in at least one position of any codeword is distinct from the symbols in the same position of any other w codewords. This is the same as that for any set of w rows A in T there exists at least one column that the entries in A are distinct from the entry in a row that is not A. $\mathcal{H}(C)$ by definition is an $\mathrm{SHF}(l; n, q, \{w, 1\})$. □

Frameproof codes are the weakest form of traitor tracing codes. Limiting the coalition size to w, a w-frameproof code does not provide any guarantee to the tracing authority that any traitor will be identified. It merely assures that no

innocent user can be incriminated. A stronger notion called a *w-secure frameproof code*, introduced by Stinson et al. [104], protects against two disjoint coalitions being able to produce a common descendant. However, a w-secure frameproof code still does not provide explicit traceability.

Definition 5.3.5. An (l, n, q)-code C is a w-secure frameproof (w-SFP) code if $x \in \mathbf{dc}(C_0) \cap \mathbf{dc}(C_1)$ implies that $C_0 \cap C_1 \neq \emptyset$ for any two subsets $C_0,\ C_1$ of C of cardinality at most w.

Thus, in a w-secure frameproof code, no coalition of size at most w can frame a disjoint coalition of size at most w by producing an l-tuple that could have been produced by the second coalition.

Just as we can construct FP codes from separating hash families, we can construct SFP codes from separating hash families in a similar way.

Theorem 5.3.2. *[104] Let $n \geq 2w$. An (l, n, q)-code, C, is a w-secure frameproof code if and only if $\mathcal{H}(C)$ is an* $\mathrm{SHF}(l; n, q, \{w, w\})$.

The proof of this theorem is similar to that of Theorem 5.3.1. Table 5.3.1 [101, Chapter 14.4.3] shows an example of a 2-SFP $(3, 7, 5)$-code and hence an $\mathrm{SHF}(3; 7, 5, \{2, 2\})$.

0	0	0
0	1	1
0	2	2
1	0	3
2	0	4
3	3	0
4	4	0

Table 5.1: A 2-SFP $(3, 7, 5)$-code

For information on constructing separating hash families, refer to [81, 104]. There are also some efficient methods for constructing 2-SFP codes [107, 108].

5.3.2 Identifiable Parent Property Codes

An identifiable parent property (IPP) code is a strong version of a tracing code. It allows at least one of the colluders to be identified, and hence codes with the IPP are deterministic. IPP codes were first studied by Hollmann et al. [64]. They study IPP codes in the case that the number of colluders is at most 2. Staddon et al. [99] later generalized the result for the case of an arbitrary number of colluders.

The idea of IPP codes is that given a hybrid fingerprint, one finds all the coalitions that are capable of generating this hybrid fingerprint. Codewords that are present in all those coalitions are called *parents* of the hybrid fingerprint, and are identified as traitors in the system.

Let $x \in \mathbf{dc}_w(C)$ be a hybrid fingerprint. The set of *suspect coalitions* of x is defined as follows:

$$\mathbf{sc}_w(x) = \{C_0 \subseteq C \mid |C_0| \leq w, x \in \mathbf{dc}(C_0)\}.$$

Therefore, $\mathbf{sc}_w(x)$ consists of all the coalitions of size at most w that are capable of generating x. If $\mathbf{sc}_w(x)$ has only one set in it, the tracing authority would incriminate this set as the guilty coalition. If $\mathbf{sc}_w(x)$ has more than one set in it, we would still want to extract some useful information from those sets.

Definition 5.3.6. An (l, n, q)-code C is a w-IPP code if for all $x \in \mathbf{dc}_w(C)$, it holds that

$$\bigcap_{C_0 \in \mathbf{sc}_w(x)} C_0 \neq \emptyset.$$

Thus, a code has the w-IPP if no coalition of size at most w can generate a hybrid fingerprint that cannot be traced back to at least one member of the coalition. Table 5.3.1 in the previous section shows a (3, 7, 5) 2-IPP code by viewing each row as a codeword. This code is also 2-SFP as shown in the previous section. A 2-IPP code is a 2-SPF code, however, the converse is generally not true. We will explore the relationships among different types of codes in Section 5.3.4.

Hollmann et al. [64] showed that an upper bound on the size of an (l, n, q) 2-IPP code is

$$n \leq 3q^{\lceil \frac{l}{3} \rceil}.$$

Hence, for a fixed alphabet size the number of codewords grows at most exponentially with the codeword length.

Staddon et al. [99] showed an upper bound on the size of an (l, n, q) w-IPP code, which is

$$n \leq q^{\lceil \frac{l}{w} \rceil} + 2w - 2.$$

This upper bound has been improved by Blackburn to

$$\frac{1}{2}u(u-1)q^{\lceil l/(u-1) \rceil},$$

where $u = \lfloor (k/2+1)^2 \rfloor$ [27].

A lemma proposed by Hollmann et al. [64] provides necessary and sufficient conditions for a code to be 2-IPP. We present this lemma as a theorem below, and we also give a proof of it.

Theorem 5.3.3. *[64] A (l, n, q)-code C has the 2-IPP property if and only if the two following conditions are satisfied:*

(1) a, b and c are distinct codewords in $C \Rightarrow a_i$, b_i and c_i are distinct in Q for some $1 \leq i \leq l$.

(2) a, b, c and $d \in C$ with $\{a, b\} \cap \{c, d\} = \emptyset \Rightarrow \{a_i, b_i\} \cap \{c_i, d_i\} = \emptyset$ for some $1 \leq i \leq l$.

Proof. We show that the two conditions are necessary for C to have 2-IPP by contradiction. Assume that C has the 2-IPP, but does not satisfy one of the two conditions. If condition (1) is not satisfied, there exist three distinct codewords a, b and c with no distinct entries on any position among them. Take the symbol on each position that is common to at least two codewords and form a hybrid

fingerprint. This hybrid fingerprint can be generated by any two of the three codewords. However, $\{a, b\} \cap \{a, c\} \cap \{b, c\} = \emptyset$, thus C does not have the 2-IPP, and we reach a contradiction. If condition (2) is not satisfied, there exist a, b, c and $d \in C$ with $\{a, b\} \cap \{c, d\} = \emptyset$ and $\{a_i, b_i\} \cap \{c_i, d_i\} = x_i$ where $x_i \in Q$ for all $1 \leq i \leq l$. Take all the x_i's and form a hybrid fingerprint x. Now we have $x \in \mathbf{dc}(a, b)$, $x \in \mathbf{dc}(c, d)$ and $\{a, b\} \cap \{c, d\} = \emptyset$. Thus C does not have the 2-IPP, and we reach a contradiction.

We show that the two conditions are sufficient for C to have 2-IPP by contrapositive. Suppose that C does not have the 2-IPP, there exists case 1: three distinct codewords a, b and $c \in C$ such that $x \in \mathbf{dc}(a, b)$, $x \in \mathbf{dc}(a, c)$ and $x \in \mathbf{dc}(b, c)$ and clearly $\{a, b\} \cap \{a, c\} \cap \{b, c\} = \emptyset$, or case 2: a, b, c and $d \in C$ with $\{a, b\} \cap \{c, d\} = \emptyset$ such that $x \in \mathbf{dc}(a, b)$ and $x \in \mathbf{dc}(c, d)$, or both case 1 and case 2. These are the only possible cases for C not to be 2-IPP. Any case with more than four distinct codewords is equivalent to case 2. In case 1, we have $x_i \in \{a_i, b_i\}$, $x_i \in \{a_i, c_i\}$ and $x_i \in \{b_i, c_i\}$, which implies that $\{a_i, b_i\} \cap \{a_i, c_i\} \cap \{b_i, c_i\} \neq \emptyset$ for all $1 \leq i \leq l$. Hence, at least two of the a_i, b_i and c_i are equal to x_i for all $1 \leq i \leq l$, and condition (1) does not hold. In case 2, we have $x_i \in \{a_i, b_i\}$ and $x_i \in \{c_i, d_i\}$ for all $1 \leq i \leq l$. Thus, $\{a_i, b_i\} \cap \{c_i, d_i\} \neq \emptyset$ for all $1 \leq i \leq l$, and condition (2) does not hold. □

The two conditions of this lemma can be translated into hash families with certain properties, and hence we can set up relations between IPP codes and hash families. Before doing so, we need to know a specific kind of hash family called *perfect hash family*.

Definition 5.3.7. Let n, q and w be positive integers and $n \geq q \geq w \geq 2$. An (n, q, w)-perfect hash family is an (n, q)-hash family, $\mathcal{H}$, such that for any $X_1 \subseteq X$ with $|X_1| = w$, there exists at least one $h \in \mathcal{H}$ such that $h(x)|_{X_1}$ is one-to-one. We use the notation PHF$(l; n, q, w)$ to denote an (n, q, w)-perfect hash family with $|\mathcal{H}| = l$.

A PHF$(l; n, q, w)$ can be depicted as an $n \times l$ table, much like we did for the

separating hash family. Each entry is an element from Y, and each column corresponds to one of the functions in $\mathcal{H}$. The table has the property that in any w rows A there exists at least one column such that the w entries in A are distinct.

The first condition of Theorem 5.3.3 corresponds to a PHF$(l; n, q, 3)$, because it fulfills the property that in any three rows in the table representing C there exists at least one column such that the three entries are distinct. Such a table is exactly a PHF$(l; n, q, 3)$. The second condition corresponds to an SHF$(l; n, q, \{2, 2\})$, because the table representing this separating hash family guarantees that for any two disjoint sets of two rows, there is a column where the entries in one set are distinct from the entries in the other. Viewing the rows as codewords, the table is a code with the second condition. Therefore, we can translate Lemma 5.3.3 into the following theorem which gives a relation between 2-IPP codes and hash families.

Theorem 5.3.4. *[99] An (l, n, q)-code C, is a 2-IPP code if and only if $\mathcal{H}(C)$ is simultaneously a PHF$(l; n, q, 3)$ and an SHF$(l; n, q, \{2, 2\})$.*

For example, Table 5.3.1 in the last section is simultaneously a PHF$(3; 7, 5, 3)$ and an SHF$(3; 7, 5, \{2, 2\})$. Hence the rows of the table forms a 2-IPP (3, 7, 5)-code.

There are many constructions of 2-IPP codes. To trace traitors, we need *tracing algorithms* for the codes. For IPP codes, we call these algorithms *parent-identifying algorithms*. Some constructions give 2-IPP codes with tracing complexity $O(1)$ [64]. The construction of w-IPP codes is a bit trickier. Staddon et al. [99] presented a construction from perfect hash families.

Theorem 5.3.5. *[99] An (l, n, q)-code C, is a w-IPP code if $\mathcal{H}(C)$ is a $PHF(l; n, q, \lfloor (w+2)^2/4 \rfloor)$.*

For a proof of this theorem, refer to [99, Theorem 2.8]. The paper also shows that a w-IPP code cannot exist on an alphabet of size less than or equal to w. This raised the question of whether there exist w-IPP codes for $w < q < \lfloor (w+2)^2/4 \rfloor$. This question was addressed in [21, 22]. The answer to the question is positive. Several explicit constructions of w-IPP codes have been suggested [20, 95].

A generic algorithm for identifying parents of a hybrid fingerprint is to intersect all the suspect coalitions of the hybrid fingerprint, which gives tracing complexity $O(\binom{n}{w})$. One recent work by van Trung and Martirosyan provides construction of a class of w-IPP codes with linear tracing complexity in the number of codewords [110]. This leads to our next topic, in which we will study a class of strong tracing codes called *traceability codes* whose tracing algorithms are linear or faster in the number of codewords.

5.3.3 Traceability Codes

Traceability (TA) codes, which are first discussed in [44, 45], allow identification of at least one of the members in a coalition. Furthermore, the tracing can be done in linear time with respect to the total number of codewords. The tracing algorithms for TA codes are based on *nearest neighbor decoding*, which is used in error-correcting codes. With a TA code, for any hybrid fingerprint generated by a coalition whose size is under a certain threshold, the codeword that is "closest" to the hybrid fingerprint is always in the coalition.

Definition 5.3.8. For any two codewords $a, b \in C$, let $\mathbf{dist}(a, b) = |\{i \mid a_i \neq b_i\}|$ denote the Hamming distance between a and b. An (l, n, q)-code C is a w-TA code if for all $C_0 \subseteq C$ with $|C_0| \leq w$ and all $x \in \mathbf{dc}(C_0)$ there exists at least one codeword $y \in C_0$ such that $\mathbf{dist}(x, y) < \mathbf{dist}(x, z)$ for all $z \in C \setminus C_0$. y is called a nearest neighbor to x and is denoted by $\mathbf{nn}(x)$.

There could be many nearest neighbors to x in C_0. Computing $\mathbf{nn}(x)$ is called nearest neighbor decoding, which can be easily done with an exhaustive search over all the codewords in C in time $O(n)$. Faster algorithms exist depending on the structure of the code [62, 98].

Chor et al. observed a sufficient condition for a code to be a w-TA code [45], which is for the code to have a large minimum distance between distinct codewords.

Theorem 5.3.6. *[45] Let* $\mathbf{dist}(C)$ *denote the minimum distance between any two codewords in* C*;* $\mathbf{dist}(C) = \min\{\mathbf{dist}(a,b) \mid a,b \in C, a \neq b\}$*. If* C *is an* (l,n,q)*-code with the property that*

$$\mathbf{dist}(C) > l\left(1 - \frac{1}{w^2}\right),$$

then C *is a* w*-TA code.*

Proof. [101, Theorem 14.6] For any two codewords $a, b \in C$, define $\mathbf{match}(a,b) = l - \mathbf{dist}(a,b)$. Suppose that x is a hybrid fingerprint generated by a coalition of size at most w, and $C_0 \in \mathbf{sc}_w(x)$. Let y be a nearest neighbor of x. We need to prove that $y \in C_0$.

Because $x \in \mathbf{dc}(C_0)$, we have that

$$\sum_{c' \in C_0} \mathbf{match}(x,c') \geq l,$$

since every entry in x is from at least one codeword in C_0.

Since $|C_0| \leq w$, there exists a codeword $c' \in C_0$ such that

$$\mathbf{match}(x,c') \geq \frac{l}{w}.$$

Because y is a nearest neighbor to x, it follows that

$$\mathbf{match}(x,y) \geq \frac{l}{w}.$$

Let $d \in C \setminus C_0$. Because $x \in \mathbf{dc}(C_0)$, we have that

$$\mathbf{match}(x,d) \leq \sum_{c' \in C_0} \mathbf{match}(c',d) \leq w(l - \mathbf{dist}(C)).$$

The condition $\mathbf{dist}(C) > l(1 - 1/w^2)$ is equivalent to

$$w(l - \mathbf{dist}(C)) < \frac{l}{w},$$

and thus $\mathbf{match}(x,d) < \mathbf{match}(x,y)$ for all codewords $d \in C \setminus C_0$. Hence, $y \in C_0$, and therefore C is a w-TA code. □

Stinson and Wei [105] gave an explicit construction of TA codes based on *orthogonal arrays* which are equivalent to *Reed-Solomon codes*. These codes are used by AACS for its traitor tracing, according to its designers [71].

Definition 5.3.9. Let q be a prime power and a positive integer $t < q$. Let the set $P(q,t)$ contain all polynomials $f(x) \in GF(q)[x]$ having degree at most $t-1$. For a positive integer $l < q$, define

$$\mathcal{RS}(q,l,t) = \{(f(0), f(1), \ldots, f(l-1)) \mid f(x) \in P(q,t)\}.$$

$\mathcal{RS}(q,l,t)$ is a Reed-Solomon code.

Theorem 5.3.7. *Let q be a prime power and choose two positive integers l and t that are smaller than q. Then, $\mathcal{RS}(q,l,t)$ is a w-TA $(l, q^{\lceil \frac{l}{w^2} \rceil}, q)$-code for $w \geq 2$.*

Proof. [99] $\mathcal{RS}(q,l,t)$ is an (l, q^t, q) code with $\mathbf{dist}(C) = l - t + 1$. This is due to the fact that any two distinct polynomials of degree at most $t-1$ defined over a field can agree on at most $t-1$ points.

Let us define

$$t = \left\lceil \frac{l}{w^2} \right\rceil$$

where $w \geq 2$, then

$$t < \frac{l}{w^2} + 1.$$

Therefore,

$$\mathbf{dist}(C) = l - t + 1 > l - \frac{l}{w^2} - 1 + 1 = l(1 - \frac{1}{w^2}).$$

Hence, the $(l, q^{\lceil \frac{l}{w^2} \rceil}, q)$-code is a w-TA code. □

In [99], an interesting problem is raised: Can we construct w-TA codes with $q < w^2$ and $n > q$? A positive answer is given by van Trung and Martirosyan [111] along with explicit constructions of such codes. There are other construction techniques for TA codes, e.g., in [80], [119].

5.3.4 Relationships Among Different Types of Codes

We have studied four types of codes that can be used for traitor tracing purposes. Frameproof and secure frameproof codes are weak tracing codes, whereas identifiable parent property and traceability codes are strong tracing codes. In this section, we explore the relations among these four types of codes.

Theorem 5.3.8. *A w-SFP code is a w-FP code.*

Proof. Let code C be a w-SFP code. We show that it is also w-FP by contrapositive. Suppose that C is not w-FP. Let x be a codeword of C such that it can be framed by a coalition C_0 of size at most w, i.e. $x \in \mathbf{dc}(C_0)$ and $x \notin C_0$. Let C_x be a coalition containing only x. We have $x \in \mathbf{dc}(C_0) \cap \mathbf{dc}(C_x)$ and $C_0 \cap C_x = \emptyset$. Hence, C is not w-SFP. □

Theorem 5.3.9. *A w-IPP code is a w-SFP code.*

Proof. Similar to the proof of Theorem 5.3.8, suppose that C is not w-SFP. There exists a hybrid fingerprint x and two disjoint coalitions C_0 and C_1 such that $x \in \mathbf{dc}(C_0) \cap \mathbf{dc}(C_1)$. Since C_0 and C_1 both belong to $\mathbf{sc}_w(x)$, and $C_0 \cap C_1 = \emptyset$, C is not w-IPP. □

Theorem 5.3.10. *A w-TA code is a w-IPP code.*

Proof. [99] Let C be a w-TA code. Suppose that $x \in \mathbf{dc}_w(C)$, then there is a subset $C_i \subseteq C$, where $|C_i| = w$, such that $x \in \mathbf{dc}(C_i)$. By the definition of a w-TA code, there exist $y \in C_i$ such that $\mathbf{dist}(x, y) \leq \mathbf{dist}(x, z)$ for all $z \in C$. To prove C is also a w-IPP code, we show that for any $C_j \subseteq C$ with $|C_j| \leq w$, $x \in \mathbf{dc}(C_j)$ implies $y \in C_j$. Suppose that there exists a C_j which does not contain y, then there is $u \in C_j$ such that $\mathbf{dist}(x, u) < \mathbf{dist}(x, y)$ by the definition of a w-TA code. This contradicts the fact that $\mathbf{dist}(x, y) \leq \mathbf{dist}(x, z)$ for all $z \in C$. □

Combining the above theorems, we get

$$w\text{-TA} \;\Rightarrow\; w\text{-IPP} \;\Rightarrow\; w\text{-SFP} \;\Rightarrow\; w\text{-FP}.$$

However, the converse is not true: w-FP does not necessarily imply w-SFP which in turn does not necessarily imply w-IPP and which in turn does not necessarily imply w-TA. Examples are easy to construct.

5.4 The AACS Traitor Tracing Scheme

AACS provides a traitor tracing mechanism which allows the AACS LA to trace the source of unauthorized copies of protected content. The AACS specifications provide little information on its traitor tracing mechanism. A handful of papers on the AACS's traitor tracing scheme, written by its designers, provide some details on its traitor tracing mechanism [66, 67, 68, 69, 70, 71, 72]. These papers are the main source from which we have obtained information on AACS's traitor tracing scheme. The various parameters used in this section to describe the AACS traitor tracing mechanism do not necessarily correspond to the parameters used for real implementations, which are confidential and implementation-specific.

The AACS traitor tracing scheme is a combinatorial traitor tracing scheme which employs TA-codes. Specifically, it is based on the Reed-Solomon codes that we studied in the last section [71]. Two Reed-Solomon codes are used in the scheme in such a way that one code is nested inside another in order to accommodate a large number of devices, while at the same time minimizing disc space overhead.

5.4.1 Inner Code

The inner code is applied to each individual movie. For each movie, l_1 different segments are chosen. Each segment is of length s seconds and is fingerprinted differently for q_1 times to produce q_1 variations. All the variations of each segment are included on the disc. A playing device can only play back the movie through a specific path, which effectively creates a unique movie version for this device or the model of devices this device belongs to. Each version of the content contains one variation for each segment. The l_1 segments form a codeword with alphabet $\mathcal{Q}_1 =$

$\{1, 2, \ldots, q_1\}$. We use C_1 to denote the set of all AACS codewords for a particular movie and l_1 to represent the length of the codeword, which is constant across movies. AACS systematically allocates the variations based on Reed-Solomon codes with dimension t_1. $C_1 = (l_1, q_1^{t_1}, q_1)$ is called the inner code.

Since we include all the variations of each segment onto the disc, the scheme introduces disc space overhead. Movie studios would want to keep this overhead low, usually below 10% [66]. AACS designers suggested to have 8 additional minutes (480 seconds) of content strictly for forensic purposes [70]. Therefore, the number of segments, variations, and the length of the segment have to be chosen in such a way that the overhead is not much more than 8 minutes, and, on the other hand, all the segments together should represent a non-negligible part of the movie. For example, if a segment is of 1/2-second duration, 960 variations could have been produced. However, attackers could simply omit that 1/2 second in the unauthorized copy, and the value of the copy would not be significantly diminished. The designers of the AACS traitor tracing scheme settled on a model where for each movie, 15 different 2-second segments are chosen, and each segment has 16 variations. The overhead introduced by this model is 450 seconds which is within the 10% constraint placed by the studio. The segments take up altogether 30 seconds of the movie. If the attacker decides to avoid them, the value of the pirated movie would be decreased. The dimension of the code is 2. Therefore $C_1 = (15, 16^2, 16) = (15, 256, 16)$. The inner code generates 256 versions of the movie. Although these parameters are the ones suggested by the designers of the AACS traitor tracing scheme, they are not necessarily the ones that will be deployed when the scheme is eventually implemented [69].

Movie segments are fingerprinted using an audio watermark technology developed by Verance Corporation [113]. This technology can embed digital codes into almost any audio recording and transmission such as music, spoken word, audio-visual material, etc. Allegedly, Verance's watermarking technology resists forgery, unauthorized alteration, decoding and erasure [114].

Using Theorem 5.3.7, we can calculate the tracing capacity of the inner code with the parameters suggested by its designers:

$$t_1 = \left\lceil \frac{l_1}{w_1^2} \right\rceil \Rightarrow 2 = \left\lceil \frac{15}{w_1^2} \right\rceil \Rightarrow w_1 = 3.$$

The inner code is 3-TA. $dist(\mathcal{C}_1) = l_1 - t_1 + 1 = 14$, which means that among the 256 different encodings of a movie, no two encodings share more than one identical (fingerprinted) segment.

There are two problems associated with the inner code: it cannot accommodate a large number of devices in the system, and it is unable to trace devices in a large coalition. Generally, for a traitor tracing scheme to be practical, we want it to have a small disc space overhead, to accommodate a large number of devices, and to effectively trace traitors in a big coalition. However, theoretically these three factors are inherently conflicting. For example, in the case of Reed-Solomon codes, to reduce disc space overhead, the codeword cannot be too long, and the size of the alphabet q_1 can not be too large. Under this constraint, if we want to accommodate a large number of devices, then the dimension of the code, t_1, has to be big. However, if t_1 is big, the size of the coalition, w_1, has to be small in order to make the code w_1-TA. The AACS attempted to solve these conflicts by introducing the *outer code*.

5.4.2 Outer Code

The outer code is applied to each playback device. It uses the codewords of the inner code as its alphabet. Each entry in the outer codeword is a version of a movie. Thus, the outer code and the inner code are interconnected. The designers have also picked a model for the outer code. The length of a codeword in the outer code, l_2, is 255. Each entry in an outer codeword is a version of a movie, and the codeword represents a sequence of 255 different movies. Since a movie has 256 encodings due to the inner code, the alphabet size of the outer code, q_2, is 256. The dimension of the outer code, t_2, is 4. Therefore, the outer code, C_2, is a $(255, 256^4, 256)$ code.

The outer code assigns different movie versions to the device over a sequence of movies. It can accommodate more than four billion devices. The inner code nests inside the outer code in an obvious way, and the two-level codes together form a *supercode*.

We now show a small example of how an outer code and an inner code are used in AACS traitor tracing. To simplify the example, suppose that an outer code has codeword length 2 and alphabet size 3. A playing device D is assigned with an outer codeword $(1, 3)$, which means that D is able to play the first version of the first movie and the third version of the second movie. The choice of the two movies in the sequence is determined by the movie studio or the AACS LA. Since each entry in an outer codeword is a codeword in the inner code, the size of the inner code has to be 3. Let the inner code have codeword length 2, alphabet size 3 and dimension 1, for example $\{(1, 2), (2, 3), (3, 1)\}$. Therefore, each movie has two segments, and each segment has 3 variations. The super codeword for D is thus:

$$((1, 2), (3, 1)).$$

This super codeword indicates that for the first movie D can play the first variation of the first segment and the second variation of the second segment, and for the second movie it can play the third variation of the first segment and the first segment of the second variation. If two pirated copies of movies are in a sequence and have fingerprints ((1, 2), (3, 1)), then D could potentially be identified as a traitor.

However, the outer code has two drawbacks. First of all, it needs 255 pirated movies in a sequence to trace the traitors. If attackers decide not to pirate some of the movies in the sequence, and mix-and-match the movies that they do pirate, it could render the tracing mechanism ineffective. This attack model has not yet been studied and therefore is a topic of future research. Second, the outer code has weak traceability. According to Theorem 5.3.7, the outer code is 9-TA:

$$t_2 = \left\lceil \frac{l_2}{w_2^2} \right\rceil \;\Rightarrow\; 4 = \left\lceil \frac{255}{w_2^2} \right\rceil \;\Rightarrow\; w_2 = 9.$$

Jin and Lotspiech [69] show the fact that the outer code is 9-TA using a counting technique, which is essentially the interpretation of $t_2 = \lceil \frac{l_2}{w_2^2} \rceil$:

"Since $dist(\mathcal{C}_2) = l_2 - t_2 + 1 = 252$ and the length of the codeword is 255, no two players follow identical paths in more than three movies (in other words, no two codewords share more than three identical positions). Now consider a coalition of w players being used in an attack and an innocent player P that, by misfortune, has maximal overlap with each player (We are assuming the attackers do not know the codes of any innocent player to deliberately incriminate it. In the AACS, the codes remain secret, and this is a reasonable assumption). P's score (the number of movies P has in common with the recovered movies) is $3w$. At least one traitor must have scored greater than or equal to $255/w$. This analysis reveals that w cannot be greater than 9 to have a deterministic certainty that the highest scoring player is a traitor rather than being innocent."

In [72], the designers claimed that they found a way to trace a decent size of coalition without the need of having all 255 pirated movies. Their technique has nothing to do with the method of constructing the codewords. It is about how the tracing is performed. This technique works well experimentally, although theoretically it does not directly apply to Reed-Solomon codes.

The technique is based on probabilistic analysis. For n players and a sequence of m pirated movies, each movie having one random version out of q, the expected number of coalitions of size w that could produce those m movies are:

$$\binom{n}{w} \times (1 - (1 - \frac{1}{q})^w)^m. \tag{5.1}$$

$1 - (1 - 1/q)^w$ is the probability that at least one member in the coalition has the same variation as the pirated movie in consideration.

If the result is less than 1, it represents the probability that this sequence of m pirated movies is covered by a coalition of size w. The intuition behind this formula is that we want the value to be very small, which implies that the probability for a coalition of size w to cover this sequence of pirated movies is very small. Therefore,

if we do find a coalition of size w, it is unlikely that this has happened by chance, and we can incriminate this coalition as the guilty coalition.

A tracing algorithm is given in [72], which basically tries all possible coalitions of size w and sees which one can cover the sequence of pirated movies. Once a coalition is found, the algorithm outputs the coalition and terminates. The algorithm is named *set-covering* algorithm because finding a coalition of devices that covers a sequence of movies resembles the well-known *set cover* problem. The algorithm is shown in Algorithm 4.

However, formula (5.1) considers the case where the entries in every codeword are assigned randomly. For Reed-Solomon codes, the number of entries in a codeword are random only up to the dimension if the code, and the rest of the entries are then totally determined. For example, in the model for the outer code, only four entries in a codeword are random. Once these four entries are chosen, the other 251 entries are determined. In addition, this tracing method does not deal with the "scapegoat strategy" in which the coalition sacrifices a few devices and uses them heavily (scapegoats) while using the others lightly. In this case, there could be many semi-innocent coalitions that can cover the sequence of pirated movie due to the fact that they contain those scapegoats. The tracing algorithm suggested in [72] can be modified to cope with this strategy: Output all the coalitions that can cover the sequence of pirated movies, not just the first one, and then intersect them to find the scapegoats.

Nevertheless, the results from our simulation have shown that this tracing method works well for Reed-Solomon codes. We implemented Algorithm 4 with a slight modification to make it capable of dealing with the scapegoat strategy. Table 5.2 shows the results of our simulation.

The Reed-Solomon code used in our simulation is $(36, 37^3, 37)$ which is a 4-TA code. The size of the guilty coalitions ranges from 4 to 7. m represents the number of pirated movies. If m is 36, the versions of the pirated movies form a hybrid fingerprint. If m is smaller than 36, then only a partial hybrid fingerprint is

Procedure `Cover`(w, *entry_list*)

3.1 **if** $w * t > m$ **then** // t is the dimension of the code.
3.2 // Too many coalitions, not traceable;
3.3 return false;
3.4 **end**
3.5 $min = \lceil \frac{m}{w} \rceil$;
3.6 **for** *each codeword in the code* **do**
3.7 **if** *this codeword covers min number of entries in the entry_list* **then**
3.8 **if** $w == 1$ **then**
3.9 print the device ID;
3.10 return true;
3.11 **else if** $w > 1$ **then**
3.12 *entry_list* = *entry_list* − all the entries covered by this codeword;
3.13 $w = w - 1$;
3.14 **if** Cover(w, *entry_list*) *returns true* **then**
3.15 print the device ID and return true;
3.16 **end**
3.17 **end**
3.18 **end**
3.19 **end**
3.20 if all the codewords have been checked, return false;

available. Given the code, the size of the coalition, and the length of the sequence of pirated movies, formula (5.1) calculates the corresponding probability that this sequence of pirated movies can be covered by a coalition of the size specified (or the expected number of coalitions if the number is greater than 1). We performed a total of 14 trials of this simulation. For each trial, we ran the tracing algorithm 10 times, each time with a different coalition and thus a different hybrid fingerprint. The coalition and the hybrid fingerprint are generated randomly. For each run

Algorithm 4: Set-Covering Algorithm

4.1 $w = 1$;

4.2 *entry_list* = the sequence of m pirated movies.

4.3 call Cover(w, *entry_list*);

4.4 **if** *procedure Cover returns true* **then**

4.5 exit;

4.6 **else**

4.7 $w = w + 1$ and loop back to line 4.3;

4.8 **end**

of the algorithm, if we can identify at least one traitor, we mark this run as a successful run. If we can identify the whole coalition, we mark this run as a unique run. A unique run is a successful run. For example, for simulation 3, there are four members in the guilty coalition, and they have collectively disseminated 18 pirated movies. Therefore, we do not have the whole hybrid fingerprint produced by the coalition, but only part of it. Formula (5.1) suggests that the probability for this sequence of entries to be covered by a coalition of size 4 is 0.54. For the entire 10 runs of the tracing algorithm, each of them has successfully identified at least one colluder. However, no run could identify the whole coalition. For simulation 5, only 3 runs have successfully identified at least one colluder. The other 7 runs have produced suspect coalitions with empty intersection, which means that the tracing has failed since we cannot identify any traitors. For simulation 1, every run has successfully identified the entire coalition.

In Table 5.2, we can see that there is a fairly strong correlation between the theoretical result given by formula (5.1) and the statistical result produced by running the tracing algorithm, even though formula (5.1) does not directly apply to the Reed-Solomon code. The statistical results indicate that we often can do better than the theoretical traceability provided by the code. The code we used in our simulation has traceability 4. However we can trace a coalition of size 6 with

Simulation Number	Code	Traceability	Size of Coalition	m	Output Given by Formula (5.1)	Results	
						successful	unique
1	$(36, 37^3, 37)$	4	4	25	6.9×10^{-8}	10/10	10/10
2	$(36, 37^3, 37)$	4	4	20	5.8×10^{-3}	10/10	8/10
3	$(36, 37^3, 37)$	4	4	18	0.54	10/10	0/10
4	$(36, 37^3, 37)$	4	4	17	5.2	7/10	2/10
5	$(36, 37^3, 37)$	4	4	16	49.8	3/10	0/10
6	$(36, 37^3, 37)$	4	5	36	2.0×10^{-11}	10/10	9/10
7	$(36, 37^3, 37)$	4	5	30	4.6×10^{-6}	10/10	4/10
8	$(36, 37^3, 37)$	4	5	25	0.13	10/10	2/10
9	$(36, 37^3, 37)$	4	5	23	8.2	5/10	0/10
10	$(36, 37^3, 37)$	4	5	21	497.7	0/10	0/10
11	$(36, 37^3, 37)$	4	6	36	7.5×10^{-5}	10/10	2/10
12	$(36, 37^3, 37)$	4	6	31	0.93	9/10	2/10
13	$(36, 37^3, 37)$	4	6	30	6.2	10/10	0/10
14	$(36, 37^3, 37)$	4	7	36	86.3	3/10	0/10

Table 5.2: Simulation results of algorithm 4

high success rate (even in cases where only partial hybrid fingerprints are available). It could be future work to modify formula (5.1) so that it could work directly for Reed-Solomon codes.

The supercode enables accommodation of a large number of devices while introducing small disc space overhead. However, it has weak traceability. Using the models suggested by the designers of the AACS traitor tracing scheme, the supercode has theoretical traceability 3, which is the traceability of the inner code. If four attackers have the same version of a movie, which is possible due to the outer code, they can mix-and-match the variations of the segments to produce an untraceable copy of the movie. Using the Internet as their communication infrastructure, pirates these days usually come in groups of a few dozens or even hundreds. The relatively

weak traceability of the AACS traitor tracing scheme could be challenged by those large groups of pirates. It is also future work for researchers to develop efficient traitor tracing schemes that can deal with pirate groups of such magnitudes.

5.5 Implementation of the AACS Traitor Tracing Scheme

Although the AACS traitor tracing scheme has not yet been implemented for HD DVD and Blu-ray discs, a few papers written by its designers present altogether a hight level view regarding its implementation [67, 69, 70, 71]. In this section, we study a few issues in the implementation of the AACS traitor tracing scheme as well as the corresponding solutions suggested in those papers.

5.5.1 Key Storage

Each segment variation is encrypted with a *variation key*. To be able to play a variation of a segment, a playing device has to have the variation key corresponding to that variation. If the inner codeword has length 15, the playing device has to have 15 variation keys in order to properly play back the movie. In addition, the outer code assigns different movie versions to the device over a sequence of movies. Hence, the number of keys that the device needs to store is the length of its corresponding supercodeword. Suppose that the length of an inner codeword is 15, and the length of an outer codeword is 255. The number of keys each device has to store is $15 \times 255 = 3825$. If each key is of size 128 bits, then the storage requirement is 61.2 KB. To store this much data securely and uniquely is very costly for the device manufacturers. Although storing 61.2 KB data securely on a device is not hard, it is the fact that the data has to be different for each device and kept secure during the manufacturing process that argues for a smaller number.

The solution that AACS employs is to add one level of indirection such that

Movie Version	Segment 1	Segment 2	...	Segment 15
S1	$E_{K_{S1}}(K_{v1})$	$E_{K_{S1}}(K_{v12})$	...	$E_{K_{S1}}(K_{v5})$
S2	$E_{K_{S2}}(K_{v7})$	$E_{K_{S2}}(K_{v3})$	...	$E_{K_{S2}}(K_{v16})$
...	...	...	...	...
S255	$E_{K_{S255}}(K_{v9})$	$E_{K_{S255}}(K_{v10})$	...	$E_{K_{S255}}(K_{v2})$
S256	$E_{K_{S256}}(K_{v15})$	$E_{K_{S256}}(K_{v8})$	...	$E_{K_{S256}}(K_{v7})$

Table 5.3: A variation key table stored on a disc (simplified)

each device only needs to store 255 keys [71]. Each of these 255 keys can get the 15 variation keys that are needed for playing back a movie from a variation key table stored on the disc. The space requirement now is only 1/15 of what was needed before. For the same example, if each key is 128 bits, the device requires storing 4.08 KB data securely and uniquely. The assignment of the 255 keys is based on the outer code of the traitor tracing scheme, where the keys and the sequence of 255 movies form a one-to-one correspondence. These keys are called *sequence keys*. Each sequence key has 256 versions, corresponding to the 256 movie versions of each movie decided by the inner code. A single device has only one version of each sequence key. A sequence key is used to obtain the 15 variation keys that are needed for playing a version of the movie. For every movie, there is a variation key table stored on the disc (Table 5.3).

A row of the variation key table contains all 15 variation keys corresponding to a version of the movie. Every key in the row is encrypted with the sequence key of that version (actually the variation keys are encrypted with an *output key* which we will see in Section 5.5.3, but for now let us just assume that they are encrypted with the sequence key). Hence, each sequence key can decrypt a row of variation keys, and these variation keys can decrypt a variation of every one of the 15 segments in the movie. For example, a device D has 255 sequence keys securely stored on a chip in the device. This sequence of keys correspond to a sequence of 255 movies. Assume that the second sequence key, K_{S2} which is of version 2, corresponds to the movie King Kong, and the variation key table stored on King Kong's disc is

Table 5.3. This sequence key in D can decrypt all the encrypted keys in the second row of the table. After decrypting all the information in the second row, D obtains 15 variation keys that allow it to play a variation of every segment in the movie King Kong. The variation key corresponding to the first segment is K_{v7}, which can decrypt the 7th variation of segment 1. For the second segment, the 3rd variation can be decrypted using K_{v3}. And for the last segment, the 16th variation can be decrypted by K_{v16}.

Although this technique slightly increases the storage associated with the content (on the disc), it substantially reduces the storage requirement on the device, which otherwise could be much more expensive. This technique does not change the basic property of the code.

5.5.2 Hybrid Tracing

The AACS traitor tracing scheme allows tracing to an individual device. However, under certain circumstances, the AACS LA might only want to trace to a certain model of devices. The reason is that a manufacturer may have made a mistake during the production of a model of devices which makes it easy for people to extract device keys from this model of devices. This attack model is usually referred to as *class attack*. Under class attacks, tracing to an individual device is made very hard due to the sheer volume of the number of compromised keys. The AACS traitor tracing scheme has a hybrid approach that allows either tracing to models or to individual devices [67], depending on the different needs at different times.

The sequence keys for a device are chosen from a sequence key table. The rows of the table represent the versions of the movies, and the columns of the table represent the movies in the sequence. For example, suppose that the outer code has codeword length 4 and alphabet size 16. Table 5.7 is a sequence key table, and it shows that device x gets sequence key (3, 12, 14, 1). In order to enable hybrid tracing, the AACS tracing scheme adopts a concept called "slots". The rows in the key table are grouped into clusters. In Table 5.5, four clusters are formed with each

cluster an aggregation of 4 rows. A slot is defined as an assignment of row clusters, with one cluster for each column. At a given column, two slots are either identical or completely disjoint. Slots can be assigned to individual models, and the keys within the clusters are assigned to the devices within the models. In effect, the outer code is now itself a two-level code. Let us name the code using clusters as alphabets the "slot code", and the other one using sequence keys as alphabets the "device code". The entire system is now a three-level code (slot code – device code – inner code).

version	movie 1	movie 2	movie 3	movie 4
1				x
2				
3	x			
4				
5				
6				
7				
8				
9				
10				
11				
12		x		
13				
14			x	
15				
16				

Table 5.4: A sequence key table. Device x: (3, 12, 14, 1)

Figure 5.5 shows a toy example of key assignment based on "slots". The first-level (cluster) code assigns slots to models X and Y with clusters as alphabets, and the second-level (device) code assigns keys to devices a and b within model X and

devices c and d within model Y. Model X has slot assignment (1, 3, 4, 1), and model Y has slot assignment (2, 4, 1, 3), where the entries in the codewords are cluster numbers. Device b gets (3, 4, 2, 1) within the slots assigned to model X, which makes its actual key assignment (3, 12, 14, 1) from the sequence key table.

	version	movie 1	movie 2	movie 3	movie 4
cluster 1	1	a X		Y	b X
	2	X		Y	X
	3	b X		c, d Y	a X
	4	X		Y	X
cluster 2	5	Y			
	6	d Y			
	7	Y			
	8	c Y			
cluster 3	9		a X		c Y
	10		X		d Y
	11		X		Y
	12		b X		Y
cluster 4	13		d Y	X	
	14		Y	b X	
	15		c Y	a X	
	16		Y	X	

Table 5.5: Key assignment using "slots" [67]

In the case of tracing only to model, the capability of tracing to devices can be disabled. This can be done by sending the same movie version to individual devices of the same model. Basically, we can choose a movie version from the slot, and its corresponding set of variation keys can be duplicated to the number of times equal to the number of keys within the slot. Each duplicated set of variation keys is encrypted with a sequence key within the slot. Hence, in the variation key table, all the devices in the same slot get the same variation keys for the movie.

Let us consider the original AACS outer code model. The outer code has alphabet size 256 which means that each sequence key comes with 256 versions. We can divide these 256 versions into 16 clusters with each cluster consisting of 16 versions. Now we can design a slot code using the Reed-Solomon code $(255, 16^4, 16)$. The resulting slot code is 9-TA and can accommodate 65536 models. We can then design a device code using the Reed-Solomon code $(255, 16^4, 16)$ which is also 9-TA and can accommodate 65536 devices. The cluster code and device code altogether provides an outer code that can accommodate 256^4 devices and is 9-TA (the same as the original outer code). However, if the cluster size is not carefully chosen, then the traceability of the code could be diminished. For example, people might want to increase the number of devices a device code can accommodate. With the original AACS outer code, let us design the slot code to be $(255, 16^3, 16)$ which is 11-TA and can accommodate 4096 models. And then we can make the device code to be $(255, 16^5, 16)$, which is able to accommodate a lot more devices (1048576, to be exact). However, the traceability drops down to 7, which makes the entire outer code 7-TA.

5.5.3 Renewable Tracing

The set of sequence keys and the set of device keys for a device are uniquely associated with each other. The compromised devices can be determined using the sequence keys if attackers do not mix-and-match in the inner code level, and then the standard broadcast encryption based revocation is employed to disable the device keys so that those compromised devices will no longer work with new movies. However, suppose that a new attack appears with new devices and new sets of device keys, and suppose that attackers have compromised their devices to such an extent that they can use the old compromised sequence keys. The newly pirated movies using the new device keys would only contain inner codewords that correspond to the old compromised sequence keys. The authority can learn nothing about the new attack, because the tracing will keep pointing back to the devices

from past attacks. To deal with such an attack model, AACS employs a scheme where the sequence keys can be revoked, in the sense that the attackers are forced to use new sequence keys in their attacks [70].

The idea is implemented through a structure called the *Sequence Key Block* (SKB). It combines traitor tracing with broadcast encryption technologies. Sets of sequence keys are assigned to individual devices out of the sequence key table as we discussed in Section 5.5.2. The key management structure is very similar to 4C Entity's Content Protection for Pre-recorded Media (CPPM), which we studied in Section 2.3. The AACS LA assigns SKBs to be used on the pre-recorded media, similar to CPPM's media key block (MKB).

Following the model that we have been using for the outer code, the sequence key table has 256 rows and 255 columns with rows representing versions of the movie and columns representing the sequence of movies. Each entry is a different sequence key. A single device has one key in each column. Thus, each device is assigned 255 sequence keys. If one key is compromised, there are many thousands of devices that will share this compromised key. Therefore, revocation of a single key is impossible. On the other hand, since no two devices have that many keys in common (recall that due to the outer code, no two devices share more than three sequence keys), even if the system has been heavily attacked and a significant fraction of the sequence keys is compromised, all innocent devices will have at least one column with an uncompromised key. By processing the SKB, all innocent devices can find a column in which they can calculate the correct answer, which is called the *output key*. However, compromised devices who have compromised keys in all columns cannot derive the output key. The output key is used to unlock one row in Table 5.3 to acquire the 15 keys that are needed for playing a version of the movie. There are many output keys, one for each version of a movie.

A column in the table corresponds to one of the 255 movies defined by the outer code. The output key of a movie is encrypted by the sequence keys in the movie's corresponding column if and only if these sequence keys are not compromised in

any device. Table 5.6 shows the structure of an SKB, which is the same as a MKB in CPPM. The first column is called the "unconditional" column which has an encryption of the output key, K_{o_i}, in every uncompromised sequence key entry. Devices that do not have compromised keys in that column immediately decrypt the output key. Other devices that have compromised keys in this column decrypt a key called a link key, K_{l_1}, which allows them to process a further column in the SKB. To process this further column these devices then have to use both their link keys and their sequence keys, K_{Sx_y}, in that column. Since processing this further column depends on the link key acquired from the previous column, it is called a "conditional" column. There might be multiple conditional columns, depending on whether there are compromised keys in the conditional column. On the other hand, there might be no conditional column in the SKB if there is no compromised key in the unconditional column.

Unconditional Column	Conditional Column	Conditional Column	...	Conditional Column
Column 1	K_{l_1}(Column 2)	K_{l_2}(Column 3)	...	$K_{l_{n-1}}$(Column n)
$K_{S1_1}(K_{l_1})$	$K_{l_1}(K_{S2_1}(K_{o_6}))$	$K_{l_2}(K_{S3_1}(K_{o_{78}}))$	...	$K_{l_{n-1}}(K_{Sn_1}(K_{o_{61}}))$
$K_{S1_2}(K_{o_2})$	$K_{l_1}(K_{S2_2}(K_{o_{122}}))$	$K_{l_2}(K_{S3_2}(K_{l_3}))$	...	$K_{l_{n-1}}(K_{Sn_2}(K_{o_{189}}))$
$K_{S1_3}(K_{l_1})$	$K_{l_1}(K_{S2_3}(K_{l_2}))$	$K_{l_2}(K_{S3_3}(K_{o_{44}}))$	...	$K_{l_{n-1}}(K_{Sn_3}(K_{o_{221}}))$
...	...	...	...	...
$K_{S1_{256}}(K_{o_{256}})$	$K_{l_1}(K_{S2_{256}}(K_{o_{33}}))$	$K_{l_2}(K_{S3_{256}}(K_{o_9}))$	...	$K_{l_{n-1}}(K_{Sn_{256}}(0))$

Table 5.6: A sample SKB

The subsequent additional conditional columns are produced in the same way as the unconditional column. The output key is encrypted by all uncompromised sequence keys in that column, and each of the resulting encrypted keys is encrypted again with the link key prepared for the previous column, so any device that has an uncompromised sequence key in that column can obtain the output key. The compromised sequence keys in that column will decrypt another link key which will be needed to process the next conditional column. Upon processing the last conditional column, all uncompromised devices should have found the output key,

Movie Version	Segment 1	Segment 2	...	Segment 15
S1	$E_{K_{o_1}}(K_{v1})$	$E_{K_{o_1}}(K_{v12})$	...	$E_{K_{o_1}}(K_{v5})$
S2	$E_{K_{o_2}}(K_{v7})$	$E_{K_{o_2}}(K_{v3})$	...	$E_{K_{o_2}}(K_{v16})$
...	...	...	...	...
S255	$E_{K_{o_{255}}}(K_{v9})$	$E_{K_{o_{255}}}(K_{v10})$	...	$E_{K_{o_{255}}}(K_{v2})$
S256	$E_{K_{o_{256}}}(K_{v15})$	$E_{K_{o_{256}}}(K_{v8})$	...	$E_{K_{o_{256}}}(K_{v7})$

Table 5.7: A variation key table stored on a disc

and all compromised devices decrypt a 0 using their link keys from the previous column and their sequence key for this column. This special value 0 will never be an SKB's correct output key, and can therefore always be taken as an indication that the device's sequence keys are revoked.

We have mentioned in Section 5.5.1 that, for simplicity purposes, all the variation keys in Table 5.3 are assumed to be encrypted with the sequence keys. In reality, the variation keys are encrypted with the output keys, as shown in Table 5.7.

The AACS LA claims that this sequence key management scheme is almost identical to that of CPPM. However, neither the CPPM's key management scheme nor the AACS sequence key management scheme is completely described in their corresponding official specifications. There is one study which strives to provide suitable key assignment strategies for CPPM [96].

Chapter 6

Conclusion and Future Work

6.1 Conclusion

This thesis studied a physical content distribution system in the context of AACS. We specifically focused on the study of three key components of a modern physical content distribution system: broadcast encryption for stateless receivers, mutual authentication with key agreement, and traitor tracing.

In Chapter 3, we studied one of the most difficult variants of the broadcast encryption scheme; when the receivers are stateless. The subset-cover framework serves as a prototype from which many revocation schemes for stateless receivers can be built. The subset difference revocation scheme is one such scheme which achieves a good balance between the client storage complexity and the complexity of the message expansion.

We also described how the SD scheme is implemented in AACS. From a theoretical point of view, AACS seems to be a robust content distribution system; however, it is widely known that AACS has failed to provide confidentiality to the movies that were distributed with it. It is apparent that AACS is weak against the known long-lived key attack. Unfortunately, such attacks are hard to thwart under the classic content distribution model. In order to protect most of the movies that

have already been released, even in the case where certain legitimate players are totally compromised, we might have to change the way content is distributed so that cryptographic measures capable of defeating the known long-lived key attack can be incorporated into the system without jeopardizing the usability of the system.

The SD scheme can be improved to the "layered SD" scheme. The motivation for this refinement is to reduce the client storage requirement while keeping other factors of the system essentially the same. It can be easily incorporated into AACS with the capability of providing backward compatibility to the devices that have already been manufactured.

The SD scheme can also be converted into a public-key SD scheme. We showed how to construct one using hierarchical identity-based encryption. The resulting public-key SD scheme has the same complexity of message expansion and a slightly larger client storage complexity, compared to that of the original SD scheme under a computational key assignment setting. The public-key SD scheme possesses compelling properties that the original SD scheme does not have, such as enabling everyone to broadcast messages and eliminating the need for the server to store secret information. It also opens up a window of opportunity to allow researchers to build features which would be hard to achieve using a symmetric-key broadcast encryption scheme.

In Chapter 4, we presented a rigorous analysis of the AACS drive-host authentication scheme. Specifically, we identified a few weaknesses present in the scheme which could lead to various attacks. It is yet to be known whether those weaknesses will lead to piracy of multimedia content. Nevertheless, we believe that it is not desirable for such a widely deployed system to employ a weak cryptographic protocol if it can be made more secure fairly easily. We proposed an improvement of the original scheme based on the well-established Station-to-Station key agreement protocol. The improved scheme provides secure mutual authentication as well as authenticated key agreement with key confirmation. We also discussed the security of the improved scheme. The improved scheme is designed with the goal of

requiring little change to be made to the original scheme, so implementation of the improved scheme is straightforward. In addition, the improved scheme requires less interaction between the drive and the host, and therefore it is more efficient than the original scheme. Furthermore, our improved scheme can be easily implemented on other content distribution systems such as CSS [51] and CPPM [1] which also use weak drive-host authentication schemes.

In Chapter 5, we studied how multimedia content can be modified to carry forensic data, and how forensic data can be used to trace compromised devices so that they can be disconnected from the system. Specifically, we studied deterministic combinatorial traitor tracing schemes using codes with certain structures to provide different traceability. Weak tracing codes, such as frameproof codes and secure frameproof codes, prevent a guilty coalition from framing an innocent user. Strong tracing codes, such as identifiable parent property codes and traceability codes, allow at least one member of the guilty coalition to be identified. Traceability codes serve as the foundation for the AACS traitor tracing scheme.

AACS is the first physical content distribution system to employ traitor tracing technology. Its traitor tracing scheme is based on Reed-Solomon codes. It uses two Reed-Solomon codes nested together to minimize disc space overhead as well as to accommodate a large number of devices. However, the theoretical traceability provided by the AACS traitor tracing scheme is weak compared to the size of today's possible pirate groups, which can consist of tens to hundreds of members. Although experiments have shown that the practical traceability may be better than the theoretical prediction, future research is needed to increase the traceability of current traitor tracing schemes in order to deal with large pirate groups.

To implement traitor tracing for physical content distribution systems, there are a few issues that need to be addressed. First, to store key information on the device securely and uniquely is expensive, so it is important to minimize the key storage requirement for the device. This issue can be solved by allowing some keys to be stored on the disc rather than on the device. Second, it is sometimes desirable

to trace to a model of devices rather than to each individual device. This issue can be addressed by partitioning the sequence key table into slots which are used to assign sequence keys to models, and the set of variation keys corresponding to a version of a movie are encrypted by all the sequence keys in a slot. The last issue concerns the ability to revoke sequence keys, which can be solved by using a key management structure such as the MKB of CPPM.

6.2 Future Work

Almost all content distribution systems are designed in the *black-box model*. In such a threat model, people assume that the attacker is restricted to observe only the input and output of the algorithm. Secret keys of a cryptographic scheme are hidden in the black-box and are never exposed. The security of DRMs designed in the black-box model depends on the strength of the cryptographic schemes it employs. If the black-box model were the case in reality, then AACS would not have failed to protect premium multimedia content. In reality, the attacker has far greater power when attacking DRMs than when attacking traditional cryptographic schemes. It is usually the case that the attacker has full control over the execution environment. Therefore, instead of breaking a cryptographic scheme, the attacker can simply extract the keys used in the scheme.

The threat model in the real world can be better described by the *white-box model*. In this model, the attacker has total visibility into the software implementation and its execution. To prevent an attacker from finding the key, the key needs to be hidden in the implementation. To improve the effectiveness of content distribution systems in terms of protecting premium multimedia content, these systems should be designed with the white-box model in mind. There has been some work done in this direction. For example, Chow et al. [46] proposed a method for protecting the key of AES in a white-box environment. The white-box implementation of AES and the rest of the software can then be obfuscated through code

transformation [48] in order to thwart reverse engineering. In general, this field is still in its infancy. Much work is required to establish techniques for implementing content distribution systems in the white-box model.

References

[1] 4C Entity LLC, "Content Protection For Prerecorded Media Specification", Revision 1.0, January 2003. 11, 18, 129

[2] 4C Entity LLC, "Content Protection For Prerecorded Media Specification, DVD Book", Revision 0.93, January 2001. 18

[3] AACS LA, "AACS Interim Adopter Agreement", February 15, 2006, http://www.aacsla.com/support/AACS_Interim_Adopter_Agrmt_070803.pdf 42

[4] AACS LA, "Advanced Access Content System (AACS) - Blu-ray Disc Pre-recorded Book", Revision 0.912, July 27, 2006, http://www.aacsla.com/specifications/AACS_Spec_BD_Prerecorded_0.912.pdf 2

[5] AACS LA, "Advanced Access Content System (AACS) - Blu-ray Disc Recordable Book", Revision 0.92, July 24, 2006, http://www.aacsla.com/specifications/AACS_Spec_BD_Recordable_0.92.pdf 2

[6] AACS LA, "Advanced Access Content System (AACS) - HD-DVD and DVD Pre-recorded Book", Revision 0.912, August 15, 2006, http://www.aacsla.com/specifications/AACS_Spec_HD_DVD_and_DVD_Prerecorded_0_912.pdf 2

[7] AACS LA, "Advanced Access Content System (AACS) - HD-DVD Recordable Book". Revision 0.912, July 25, 2006, http://www.aacsla.com/specifications/AACS_Spec_HD_DVD_Recordable_0.921_20060725.pdf 2

[8] AACS LA, "Advanced Access Content System (AACS) - Introduction and Common Cryptographic Elements", Revision 0.91, February 17, 2006. http://www.aacsla.com/specifications/specs091/AACS_Spec_Common_0.91.pdf 2, 39, 67, 73

[9] AACS LA, "Advanced Access Content System (AACS) - Pre-recorded Video Book", Revision 0.91, February 17, 2006, http://www.aacsla.com/specifications/specs091/AACS_Spec_Prerecorded_0.91.pdf 2, 41

[10] AACS LA, "Advanced Access Content System (AACS) - Recordable Video Book", Revision 0.91, February 17, 2006, http://www.aacsla.com/specifications/specs091/AACS_Spec_Recordable_0.91.pdf 2

[11] AACS LA, News Release, Accessed April 21, 2008. http://www.aacsla.com/news 42

[12] M. Abdalla, A. W. Dent, J. Malone-Lee, G. Neven, D. H. Phan, and N. P. Smart, "Identity-Based Traitor Tracing", PKC'07, LNCS, Vol. 4450, pp. 458-476, 2007. 95

[13] A. Adelsbach, S. Katzenbeisser, and H. Veith, "Watermarking Schemes Provably Secure Against Copy and Ambiguity Attacks", Proceedings of the 3rd ACM Workshop on Digital Rights Management, pp. 111-119, 2003. 94

[14] A. Adelsbach and J. Schwenk, "Key-Assignment Strategies for CPPM", Proceedings of the 2004 Workshop on Multimedia and Security, pp. 107-115, Magdeburg, Germany, September 2004. ix, 14, 51, 126

[15] B. B. Amberker, P. Koulgi, and M. B. Nirmala, "Some Implementation Issues in the Fiat-Naor Broadcast Encryption Schemes", International Conference on Advanced Computing and Communications, 2006. 22

[16] arnezami, "Processing Key, Media Key and Volume ID found!!!", Doom9's Forum. Accessed April 21, 2008. http://forum.doom9.org/showthread.php?t=121866&page=6 42

[17] J. Baek, K. Kim, "Remarks on the Unknown Key Share Attacks", IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, Vol. E83-A, No. 12, pp. 2766-2769, 2000. 73

[18] J. Baek, J. Newmarch, R. Safavi-Naini, and W. Susilo, "A Survey of Identity-Based Cryptography", AUUG, pp. 95-102, Victoria, Australia, 2004. 51

[19] E. Bangeman, "Blu-ray DRM Crown Jewel Tarnished with Crack of BD+", Ars Technica, November 08, 2007. `http://arstechnica.com/news.ars/post/20071108-blu-rays-drm-crown-jewel-tarnished-with-crack-of-bd.html` 45

[20] A. Barg, G. R. Blakley, and G. Kabatiansky, "Digital Fingerprinting Codes: Problem Statements, Constructions, Identification of Traitors", IEEE Transactions on Information Theory, Vol. 49, Issue 4, pp. 852-865, 2003. 104

[21] A. Barg, G. Cohen, S. Encheva, G. Kabatiansky, and G. Zémor, "A Hypergraph Approach to the Identifying Parent Property: The Case of Multiple Parents", SIAM Journal of Discrete Mathematics, Vol. 14, pp. 423-431, 2001. 104

[22] A. Barg and G. Kabatiansky, "A Class of I.P.P. Codes with Efficient Identification", Journal of Complexity, Vol. 20, Issue 2-3, pp. 137-147, 2004. 104

[23] M. Bellare and P. Rogaway, "Entity Authentication and Key Distribution", CRYPTO'93, LNCS, Vol. 773, pp. 232-249, 1994. 72

[24] M. A. Bender and M. Farach-Colton, "The LCA Problem Revisited", LATIN 2000: Theoretical Informatics, LNCS, Vol. 1776, pp. 88-94, 2000. 28

[25] O. Berkman, M. Parnas, and J. Sgall. "Efficient Dynamic Traitor Tracing", Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms, pp 586-595, 2000. 95

[26] BD+ Technologies LLC news, June 28, 2006. http://www.bdplusllc.com/news 45

[27] S. R. Blackburn, "An Upper Bound on the Size of a Code with the k-Identifiable Parent Property", Journal of Combinatorial Theory Series A, Vol. 102, Issue 1, pp. 179-185, 2003. 102

[28] S. R. Blackburn, "Frameproof Codes", SIAM Journal on Discrete Mathematics, Vol. 16, Number 3, pp. 499-510, 2003. 98

[29] G. R. Blakley, C. Meadows, and G. B. Purdy, "Fingerprinting Long Forgiving Messages", CRYPTO'85, LNCS, Vol. 218, pp. 180-189, 1985. 93

[30] S. Blake-Wilson, D. Johnson, and A. Menezes, "Key Agreement Protocols and Their Security Analysis", Proceedings of the Sixth IMA International Conference on Cryptography and Coding, LNCS, Vol. 1355, pp. 30-45, 1997. 72

[31] S. Blake-Wilson and A. Menezes, "Unknown Key-Share Attacks on the Station-to-Station (STS) Protocol", Proceedings of PKC 99, LNCS Vol. 1560, pp. 154-170, 1999. 73

[32] D. Boneh and X. Boyen, "Efficient Selective-ID Identity Based Encryption Without Random Oracles", EUROCRYPT 2004, LNCS, Vol. 3027, pp. 223-238, 2004. 53, 55

[33] D. Boneh, X. Boyen, and E-J Goh, "Hierarchical Identity Based Encryption with Constant Size Ciphertext", EUROCRYPT 2005, LNCS, Vol. 3494, pp. 440-456, 2005. 53, 55, 57

[34] D. Boneh and M. Franklin, "Identity Based Encryption from the Weil Pairing", CRYPTO '01, LNCS 2139, pp. 213-229, 2001. 51, 52, 62

[35] D. Boneh and M. K. Franklin, "An Efficient Public Key Traitor Tracing Scheme", CRYPTO'99, LNCS, Vol. 1666, pp. 338-353, 1999. 95

[36] D. Boneh, C. Gentry, B. Waters, "Collusion Resistant Broadcast Encryption with Short Ciphertexts and Private Keys", CRYPTO '05, LNCS, Vol. 3621, pp. 258-275, 2005. 23

[37] D. Boneh, A. Sahai, B. Waters, "Fully Collusion Resistant Traitor Tracing with Short Ciphertext and Private Keys", EUROCRYPT'06, LNCS, Vol. 4004, pp. 537-592, 2006. 90, 95

[38] D. Boneh and J. Shaw, "Collusion-Secure Fingerprinting for Digital Data", CRYPTO'95, LNCS, Vol. 963, pp. 452-465, 1995. 97, 98

[39] D. Boneh and B. Waters, "A Fully Collusion Resistant Broadcast, Trace, and Revoke System", Proceedings of ACM Conference on Computer and Communications Security, pp. 211-220, 2006. 90, 95

[40] M. Bosi, "Digital Rights Management Systems", Multimedia Security Technologies for Digital Rights Management, Chapter 2, pp. 23-49, 2006. 5

[41] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, and B. Pinkas, "Multicast Security: A Taxonomy and Some Efficient Constructions", Proceedings of INFOCOM '99, Vol. 2, pp. 708-716, New York, NY, March 1999. 22

[42] R. Canetti, S. Halevi, J. Katz, "A Forward-Secure Public-Key Encryption Scheme", EUROCRYPT '03, LNCS, Vol. 2656, pp. 255-271, 2003. 44

[43] R. Canetti, T. Malkin, K. Nissim, "Efficient Communication-Storage Tradeoffs for Multicast Encryption", EUROCRYPT '99, LNCS, Vol. 1592, pp. 459-474, 1999. 22

[44] B. Chor, A. Fiat, and M. Naor, "Tracing Traitors", CRYPTO'94, LNCS, Vol. 839, pp 480-491, 1994. 90, 95, 96, 105

[45] B. Chor, A. Fiat, M. Naor, and B. Pinkas, "Tracing Traitors", IEEE Transactions on Information Theory, Vol. 46, pp. 893-910, 2000. 90, 95, 96, 105, 106

[46] S. Chow, P. Eisen, H. Johnson, and P.C. van Oorschot, "White-Box Cryptography and an AES Implementation", Proceedings of the Ninth Workshop on Selected Areas in Cryptography, LNCS, Vol. 2595, pp. 250-270, 2003. 130

[47] C. Cocks, "An Identity Based Encryption Scheme Based On Quadratic Residues", Cryptography and Coding, LNCS 2260, pp. 360-363, 2001.

[48] C. Collberg, C. Thomborson, and D. Low, "A Taxonomy of Obfuscating Transformations", Technical Report 148, Department of Computer Science, University of Auckland, 1997. 131

[49] I. Cox, J. Killian, T. Leighton, and T. Shamoon, "Secure Spread Spectrum Watermarking for Multimedia", IEEE Transactions on Image Processing, Vol. 6, Issue 12, pp. 1673-1687, 1997. 94

[50] I. Cox, M. Miller, J. Bloom, and M. Miller, "Digital Watermarking", Morgan Kaufmann, 2001. 94

[51] DVD Copy Control Association, "CSS Procedural Specification", Version 2.9, January 2007. 8, 11, 129

[52] P. D'Arco and D. R. Stinson, "Fault Tolerant and Distributed Broadcast Encryption", CT-RSA 2003, LNCS, Vol. 2612, pp. 263-280, 2003. 22, 50

[53] W. Diffie, P. C. van Oorschot, and M. J. Wiener, "Authentication and Authenticated Key Exchanges", Designs, Codes and Cryptography, Vol. 2, Issue 2, pp. 107-125, Kluwer Academic Publishers, 1992. 43, 78

[54] Y. Dodis and N. Fazio, "Public Key Broadcast Encryption for Stateless Receivers", Digital Right Management '02, LNCS, Vol. 2696, pp. 61-80, 2002. 23, 57

[55] Y. Dodis and N. Fazio, "Public Key Trace and Revoke Scheme Secure Against Adaptive Chosen Ciphertext Attack", PKC'03, LNCS, Vol. 2567, pp. 100-115, 2003. 23, 95

[56] C. Ellison and B. Schneier, "Ten Risks of PKI: What You're Not Being Told About Public Key Infrastructure", Computer Security Journal, Vol. 16, No. 1, pp. 1-7, 2000. 51

[57] A. Fiat and M. Naor, "Broadcast Encryption", CRYPTO '93, LNCS, Vol. 773, pp. 480-491, 1994. 21

[58] A. Fiat and T. Tassa, "Dynamic Traitor Tracing", CRYPTO'99, LNCS, Vol. 1666, pp. 354-371, 1999. 90, 95

[59] E. Gafni, J. Staddon, and Y. L. Yin, "Efficient Methods for Integrating Traceability and Broadcast Encryption", CRYPTO'99, LNCS, Vol. 1666, pp. 372-387, 1999. 95

[60] C. Gentry and A. Silverberg, "Hierarchical ID-Based Cryptography", ASIACRYPT 2002, LNCS, Vol. 2501, pp. 548-566, 2002. 53, 105

[61] E. Gruenwedel, "Blu-ray's Copy-Protection Advantage", Home Media Magazine, Questex Publication, July 8-14, 2007. 45

[62] V. Guruswami and M. Sudan, "Improved Decoding of Reed-Solomon and Algebraic-Geometry Codes", IEEE Transactions on Information Theory, Vol. 45, Issue 6, pp. 17571767, 1999. 53, 105

[63] D. Halevy and A. Shamir, "The LSD Broadcast Encryption Scheme", CRYPTO '02, LNCS, Vol. 2442, pp. 47-60, 2002. 23, 46, 48

[64] H. D. L. Hollmann, J. H. van Lint, J. Linnartz, and L. M. G. M. Tolhuizen, "On Codes with the Identifiable Parent Property", Journal of Combinatorial Theory Series A, Vol. 82, pp. 121-133, 1998. 90, 95, 101, 102, 104

[65] F. K. Hwang and D. S. Richards, "Steiner Tree Problems", Networks, Vol. 22, Issue 1, pp. 55-89, 1992. 28

[66] H. Jin and J. Lotspiech, "Attacks and Forensic Analysis for Multimedia Content Protection", IEEE International Conference on Multimedia and Expo, pp. 1392-1395, Amsterdam, Netherlands, 2005. 109, 110

[67] H. Jin and J. Lotspiech, "Hybrid Traitor Tracing", IEEE International Conference on Multimedia and Expo, pp. 1329-1332, Toronto, Canada, 2006. viii, 109, 118, 120, 122

[68] H. Jin and J. Lotspiech, "Practical Forensic Analysis in Advanced Access Content System", Information Security Practice and Experience 2006, LNCS, Vol. 3903, pp. 302-313, Hangzhou, China, 2006. 109

[69] H. Jin and J. Lotspiech, "Practical Traitor Tracing", Multimedia Security Technologies for Digital Rights Management, Academic Press, Chapter 12, pp. 323-347, 2006. 109, 110, 112, 118

[70] H. Jin and J. Lotspiech, "Renewable Traitor Tracing: A Trace-Revoke-Trace System For Anonymous Attack", ESORICS 2007, LNCS, Vol. 4734, pp. 563 - 577, 2007. 109, 110, 118, 124

[71] H. Jin and J. Lotspiech, "Traitor Tracing for Prerecorded and Recordable Media", ACM DRM Workshop, pp. 83-90, Washington, D.C., 2004. 107, 109, 118, 119

[72] H. Jin, J. Lotspiech, and M. Blaum, "Efficient Traitor Tracing", International Symposium on Communication Theory and Applications, pp. 200-204, 2003. 109, 113, 114

[73] B. S. Kaliski Jr., "An Unknown Key-Share Attack on the MQV Key Agreement Protocol", ACM Transactions on Information and System Security, Vol. 4, No. 3, pp. 275-288, 2001. 73

[74] S. Katzenbeisser and F. Peticolas (Editors), "Information Hiding Techniques for Steganography and Digital Watermarking", Artech House Publishers, 2000. 93

[75] G. Kesden, "Content Scrambling System (CSS)", Carnegie Mellon University, 2000. http://www.cs.cmu.edu/~dst/DeCSS/Kesden/index.html 10

[76] A. Kiayias and M. Yung, "Traitor Tracing with Constant Transmission Rate", EUROCRYPT'02, LNCS, Vol. 2332, pp. 450-465, 2002. 95

[77] H. Krawczyk, "SIGMA: The 'SIGn-and-MAc' Approach to Authenticated Diffie-Hellman and Its Use in the IKE Protocols", CRYPTO'03, LNCS, Vol. 2729, pp. 400-425, 2003. 88

[78] K. Kurosawa and Y. Desmedt, "Optimum Traitor Tracing and Asymmetric Schemes", EUROCRYPT'98, LNCS, Vol. 1403, pp.145-157, 1998. 95

[79] J. W. Lee, Y. H. Hwang, and P. J. Lee, "Efficient Public Key Broadcast Encryption Using Identifier of Receivers", International Conference on Information Security Practice and Experience, LNCS, Vol. 3903, pp. 153-164, 2006. 23

[80] Y. T. Lin, J. L. Wu, and C. H. Huang, "Concatenated Construction of Traceability Codes for Multimedia Fingerprinting", Optical Engineering, Vol. 46, Issue 10, 2007. 107

[81] L. Liu and H. Shen, "Explicit Constructions of Separating Hash Families from Algebraic Curves over Finite Fields", Designs, Codes and Cryptography, Vol. 41, Issue 2, pp. 221-233, 2006. 100

[82] J. Lotspiech, "Broadcast Encryption", *Multimedia Security Technologies for Digital Rights Management*, Academic Press, Chapter 12, 2006. 39

[83] T. Martin, "A Set Theoretic Approach to Broadcast Encryption", *Ph.D Thesis*, Royal Holloway University of London, 2005. 27

[84] D. McGrew and A. T. Sherman, "Key Establishment in Large Dynamic Groups Using One-Way Function Trees", IEEE Transactions on Software Engineering, Vol. 29, Issue 5, pp. 444-458, May 2003. 22

[85] M. L. Miller, I. J. Cox, J. M. G. Linnartz, and T. Kalker, "A Review of Watermarking Principles and Practices", Digital Signal Processing in Multimedia Systems, IEEE, 461-485, 1999. 94

[86] S. Mitsunari, R. Sakai, and M. Kasahara, "A New Traitor Tracing", IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, Vol. E85-A, No. 2, pp. 481-488, 2002. 95

[87] Muslix64, "BackupHDDVD, A Tool to Decrypt AACS Protected Movies", Doom9's Forum. `http://forum.doom9.org/showthread.php?t=119871` 41

[88] National Institute of Standards and Technology, "Special Publication 800-56A, Recommendation for Pair-Wise Key Establish Schemes Using Discrete Logarithm Cryptography", March 2007. 70

[89] D. Naor, M. Naor, and J. Lotspiech, "Revocation and Tracing Schemes for Stateless Receivers", CRYPTO '01, LNCS, Vol. 2139, pp. 41-62, 2001. 22, 23, 27, 50, 90, 95

[90] M. Naor and B. Pinkas, "Efficient Trace and Revoke Schemes", FC 2000, LNCS, Vol. 1962, pp. 1-20, 2000. 23, 90, 95

[91] M. Naor and B. Pinkas, "Threshold Traitor Tracing", CRYPTO'98, LNCS, Vol. 1462, pp. 502-517, 1998. 95

[92] D. Park, C. Boyd, and S. Moon, "Forward Secrecy and Its Application to Future Mobile Communications Security", PKC 2000, LNCS, Vol. 1751, pp. 433-445, 2000. 44

[93] R. Safani-Naini and Y. Wang, "Sequential Traitor Tracing", IEEE Transactions on Information Theory, Vol. 49, pp. 1319-1326, 2003. 90, 95, 96

[94] P. Sarkar and S. Chatterjee, "Construction of a Hybrid HIBE Protocol Secure Against Adaptive Attacks". Provable Security, LNCS, Vol. 4784, pp. 51-67, 2007. 54

[95] P. Sarkar and D. R. Stinson, "Frameproof and IPP Codes", INDOCRYPT 2001, LNCS, Vol. 2247, pp. 117-126, 2001. 90, 95, 98, 104

[96] A. Shamir, "Identity-Based Cryptosystems and Signature Schemes", CRYPTO '84, LNCS 196, pp. 47-53, 1984. ix, 14, 51, 126

[97] K. Shim, "Unknown Key-Share Attack on Authenticated Multiple-Key Agreement Protocol", Electronics Letters, Vol. 39, Issue 1, pp. 38-39, 2003. 73

[98] A. Silverberg, J. N. Staddon, J. L. Walker, " Applications of List Decoding to Tracing Traitors", IEEE Transactions on Information Theory, Vol. 49, Issue 5, 2003. 105

[99] J. N. Staddon, D. R. Stinson, and R. Wei, "Combinatorial Properties of Frameproof and Traceability Codes", IEEE Transactions on Information Security, Vol. 47, Issue 3, pp. 1042-1049, 2001. 90, 95, 96, 98, 101, 102, 104, 107, 108

[100] F. A. Stevenson, "Cryptanalysis of Contents Scrambling System", November 1999. `http://www.cs.cmu.edu/~dst/DeCSS/FrankStevenson/analysis.ps` 10

[101] D. R. Stinson, "Cryptography Theory and Practice, Third Edition", Chapman & Hall/CRC, 2006. 51, 65, 66, 78, 84, 100, 106

[102] D. R. Stinson, "On Some Methods for Unconditionally Secure Key Distribution and Broadcast Encryption", Designs, Codes and Cryptography, Vol. 12, No. 3, pp. 215-243, 1997. 22

[103] D. R. Stinson and Tran van Trung, "Some New Results on Key Distribution Patterns and Broadcast Encryption", Designs, Codes and Cryptography, Vol. 14, No. 3, pp. 261-279, 1998. 22

[104] D. R. Stinson, Tran van Trung, R. Wei, "Secure Frameproof Codes, Key Distribution Patterns, Group Testing Algorithms, and Related Structures",

Journal of Statistical Planning and Inference, Vol. 86, pp. 595-617, 2000. 95, 98, 99, 100

[105] D. R. Stinson and R. Wei, "Combinatorial Properties and Constructions of Traceability Schemes and Frameproof codes", SIAM Journal on Discrete Mathematics, Vol. 11, Number 1, pp 41-53, 1998. 90, 95, 96, 98, 107

[106] D. R. Stinson and R. Wei, "Key Preassigned Traceability Schemes for Broadcast Encryption", SAC'98, LNCS, Vol 1556, pp. 144-156, 1998. 95

[107] V. D. Tô, R. Safavi-Naini, and Y. Wang, "A 2-Secure Code with Efficient Tracing Algorithm", Proceedings of the Third International Conference on Cryptology: Progress in Cryptology, LNCS, Vol. 2551, pp. 149-162, 2002. 100

[108] D. Tonien, R. Safavi-Naini, "Explicit Construction of Secure Frameproof Codes", International Journal of Pure and Applied Mathematics, Vol. 6, pp. 343-360, 2003. 100

[109] D. S. Touretzky, "Gallery of CSS Descramblers", `http://www.cs.cmu.edu/~dst/DeCSS/Gallery/` 10

[110] Tran van Trung and S. Martirosyan, "New Constructions for IPP Codes", Designs, Codes and Cryptography, Vol. 35, Issue 2, pp. 227-239, 2005. 105

[111] Tran van Trung and S. Martirosyan, "On a Class of Traceabbility Codes", Designs, Codes and Cryptography, Vol. 31, Issue 2, pp.125-132, 2004. 107

[112] A. Vampire, "WinDVD 8 Device Key Found!", Doom9's Forum. `http://forum.doom9.org/showthread.php?t=122664` 42

[113] Verance, "Verance Announces Availability of Audio Watermark Technology for High-Definition Entertainment Formats", July 2, 2007. `http://www.verance.com/news/releases/Verance_License_Announcement.pdf` 110

[114] Verance, "How Does Verance's VCMS/AV Watermarking Technology Help You Manage Your Content?". http://www.verance.com/vcms/howitworks.html 110

[115] N. R. Wagner, "Fingerprinting", Proceedings of the 1983 Symposium on Security and Privacy, pp. 18-22, 1983. 91

[116] D. M. Wallner, E. J. Harder, and R. C. Agee, "Key Management for Multicast: Issues and Architectures", Internet Request for Comments 2627, June 1999. 22

[117] B. Waters, "Efficient Identity-Based Encryption Without Random Oracles", EUROCRYPT 2005, LNCS, Vol. 3494, pp. 114-127, 2005. 23, 54

[118] C. K. Wong, M. Gouda, and S. Lam, "Secure Group Communications Using Key Graphs", IEEE/ACM Transactions on Networking, Vol. 8, Issue 1, pp. 16-30, February 2000. 22

[119] H. Yagi, T. Matsushima, and S. Hirasawa, "New Traceability Codes Against a Generalized Collusion Attack for Digital Fingerprinting", Information Security Applications, LNCS, Vol. 4298, pp. 252-266, 2007. 107

[120] D. Yao, N. Fazio, Y. Dodis, A. Lysyanskaya, "ID-Based Encryption for Complex Hierarchies with Applications to Forward Security and Broadcast Encryption", Proceedings of the ACM Conference on Computer and Communications Security, pp. 354-363, 2004. 44, 62

[121] H. Zhou, L. Fan, and J. Li, "Remarks on Unknown Key-Share Attack on Authenticated Multiple-Key Agreement Protocol", Electronics Letters, Vol. 39, Issue 17, pp. 1248-1249, 2003. 73