The Content Scramble System – A Study

https://github.com/helo2804/css_study

2015-05-18: Version 1 (this is still a draft, no reviews yet)

This document was created with LibreOffice (version 4.3.3.2).

Motivation

The playback of a video DVD is obstructed by the Content Scramble System (CSS). Its details are only given to licensees for a fee. The license itself is bound to a non-disclosure agreement. Thus, based on a CSS license, it wouldn't be possible to develop an open-source software player. However, open-source is a vital requirement for trusted software.

Luckily there are people who addressed this issue. People who re-engineered software, published the mechanism, analyzed the algorithms and revealed their weaknesses. In the end there were the developers of libdvdcss who put all this together and made their software available on many computer platforms.

Despite the success of libdvdcss, the CSS documentation remained rather poor. This project is intended to close the gap. It provides a detailed description of the CSS mechanism. If public sources are available they are cited. In fact the source-code of libdvdcss is invaluable. There are also secondary sources which may help understand CSS.

Alongside, this project is accompanied with a piece of software that is used to run some tests to verify CSS details.

Table of Contents

1 Introduction

The content scramble system (CSS) is a collection of proprietary protection mechanisms used on a DVD[1]. CSS attempts to restrict access to the content only for licensed applications. According to the DVD Copy Control Association[2] (CCA), which is the consortium that grants licenses, CSS is supposed to protect the intellectual property rights of the content owner.

From a technical point of view, CSS deals with three participants: the DVD, the drive and the player. The DVD holds the copyright information and the encrypted feature. The drive provides all means to access the DVD. The player reads, decrypts and presents the audio and visual content of the feature. All participants must be licensed and must conform to the license agreement.

The CSS implementation itself is based on three methods of protection: playback protection, copy protection and regional restriction. Playback protection is based on encryption: the player uses a secret key (or a set of secret keys[3]) to decrypt the feature. Copy protection is based on the drive: access to the DVD is only granted if the player authenticates successfully. Regional restriction is based on the DVD and the drive: access to the DVD is only granted by the drive to the player if the DVD belongs to the drive's geographic region.

For playback protection, the feature is encrypted. Three different kinds of keys are required to decrypt the feature: the player-key, which is stored in the player; and the disc- and title-keys, both stored on the DVD as cipher text.

The player-key is used to decrypt the disc-key. The disc-key in turn is used to decrypt the title-keys. Finally, the title-keys are used to decrypt the feature. The player has to perform a handshake with the drive to retrieve the encrypted disc- and title-keys from the DVD.

Each DVD-release uses its own disc- and title-keys. Hence, revealing these keys would not effect other DVD releases. However, the player-key is kind of a master key. If the player-key is revealed, the playback protection will be broken for all DVDs. That's why there exist multiple player-keys. Each DVD holds 408 encrypted variants of the disc-key. Each of them can be assigned to a dedicated player-key. Different player-keys can be assigned to different player vendors. If a certain player-key is revealed, future DVD-releases might not provide an encrypted disc-key-variant for the revealed player-key any longer.

For copy protection, the drive locks a set of disc-blocks of the feature. An attempt of the player to read the locked blocks results in an access error. To unlock the blocks, a handshake between player and drive needs to be performed.

The regional restriction is based on the region-code. Both, the DVD and the drive hold their own region-code. If the drive detects a DVD with a different region-code, the player might still be permitted to read the feature. However, the drive does not permit access to the title-keys; which are required to decrypt the feature. Still, it's possible for a consumer to change the drive's region-code manually for up to five times.[4]

1 http://en.wikipedia.org/wiki/DVD-Video

2 http://www.dvdcca.org

3 MMC[8] section 4.10.2.1

4 Rather four times. Brand-new drives hold no region-code. So the first region-code setup is performed by (most) players automatically when the first DVD is played.

All three protection methods have been virtually broken:

All it requires to circumvent the copy protection (without any special tools) is a licensed DVD player. Immediately at start-up, the player performs a handshake with the drive, and so unlocks the DVD for all applications on the PC. After that, any application can read the encrypted feature and save a copy.

The authentication handshake to unlock the DVD and to retrieve the disc- and title-keys was published in 1999 by Derek Fawcus[1]. The handshake doesn't depend on any keys and is the same for every drive.

Derek Fawcus also published the CSS playback encryption system. Its cipher uses only 40-bit keys. A brute-force attack was already feasible the date CSS was introduced. Major flaws in the cipher were revealed by Frank A. Stevenson[2] within a few days in the end of 1999. So it became possible for everyone to generate a list of valid player-keys within a few minutes on a consumer PC.

In case of regional restrictions, the drive refuses access to the title-keys; which are required to decrypt the feature. However, Ethan Hawke[3] discovered a flaw in the feature encryption which makes it easily possible to predict plain-text data. This enables the recovery of most title-keys in a fraction of a second.

For many drives there is also the option to flash another firmware with a degraded set of regional protection. If this still doesn't work, there is always the option to purchase a drive for the desired region. Only a few regions exist.

Note that a DVD can be produced with or without CSS. The publisher may for instance decide to go without CSS protection to save production costs. The situation is not that simple for the drive manufacturers: consumers who buy a drive without CSS might fear to be unable to play their DVD videos. That would make it difficult to sell such drives. Besides, the CCA is eagerly interested that all drives implement CSS. Otherwise CSS would render mostly useless. As of today, it appears that all consumer drives implement some kind of CSS protection. However, one could try to modify a consumer drive in a way that it provides raw access.[5]

5 http://debugmo.de/2007/07/read-your-dvds-the-raw-way

2 DVD-Video

DVD-Video is a proprietary video format to play movies. The official format specification is not publicly accessible. It is controlled by the DVD-Forum consortium[6] and can be bought for a license fee. The license is attached to a non-disclosure agreement. That makes it pointless to purchase a license for the sake of documentation. Luckily there are other sources.

2.1 DVD-ROM

DVD-ROM is the basic technology for DVD-Video. This chapter summarizes briefly a view details of the DVD-ROM specification[7] that might be relevant to understand the subsequent text.

1. The information on a disc-layer is stored as a sequence of pits and lands on a single path. The path is the 360°-turn of a spiral from the disc's center to the disc's rim. A bit-stream is generated through the transition from land to pit and pit to land. The bit-stream carries the payload alongside with self-clocking- and error-correction-code. A DVD may hold up to four layers.

2. The DVD's payload is provided in (logical) data-frames. Each frame holds a 10_H-byte header plus 800_H bytes of main-data. The first $30{,}000_H$ frames form the lead-in-zone. Subsequent frames form the data-zone. At the end follows the lead-out-zone.

3. The lead-in-zone holds, besides others, control-data-blocks. There is one frame for physical format information, one frame for disk manufacturing information and 14 frames for content provider information.

4. The data-frame's main-data in the data-zone are the user-data. These are logical-blocks of 800_H bytes, starting with the logical-block-address (LBA) 0 (zero) at data-frame $30{,}000_H$.

2.2 Copyright Information

The copyright information[8] is part of the content provider information in the lead-in-zone. There are two fields. The first field indicates whether CSS is applied to this DVD-Video, or not. The second field holds the region-code for this DVD-Video.

2.3 Region-Code

The 8-bit region-management-information[8] (region-code) describes the geographic regions in that a DVD-Video may be played. Each bit represents one of eight regions. If a bit is not set, the DVD-Video is permitted to be played in the corresponding region. Hence, if a bit is set, the DVD-Video is forbidden to be played in the corresponding region.

6 http://www.dvdforum.org

2.4 Disc-Key-Block

The content provider information[8] in the lead-in zone contains an 800_H-byte block with encrypted disc-keys, if the DVD is CSS-encrypted.

There are 409 5-byte slots. The first slot holds the hash-value of the disc-key. The next 408 slots are player-key-encrypted disc-key-variants. The remaining three bytes at the end are not used.[4]

The hash-value is used to match the genuine disc-key.
That is: (*k* , *c*) → *k*

k: 5-byte genuine disc-key

c: 5-byte hash-value (cipher)

So, the decryption↳4.3 of a genuine disc-key with the hash-value results in the disc-key itself.

An application, that possesses a player-key, decrypts↳4.3 successively all the 408 player-key-encrypted disc-key-variants. If a decrypted disc-key-variant matches the hash-value, the decrypted disc-key-variant is the genuine disc-key.

2.5 Copyright Management Information

The header of each data-frame on the main-data carries, besides others, one byte with copyright management information (CPR_MAI). This byte is defined as follows:[7],[8]

Field	Mask	Description
CPM	80_H	00_H: not copyrighted (fields below are not used) 80_H: copyrighted (see fields CP_SEC and CGMS)
CP_SEC	40_H	00_H: field CP_MOD is not used 40_H: field CP_MOD is used
CGMS	30_H	00_H: copying is permitted without restriction 10_H: reserved 20_H: one generation of copies may be made 30_H: no copying is allowed
CP_MOD	$0F_H$	00_H: the sector is scrambled by CSS with a title-key

2.6 Title-Key

The header of each data-frame of the main-data carries, besides others, an encrypted 5-byte title-key, if the main-data is CSS-encrypted.[7],[8] The title-key needs to be decrypted↳4.4 with the disc-key.

2.7 File-System

The universal-disk-format (UDF) version 1.02 serves as DVD file-system. The user-data's logical-blocks are the file system's building blocks. Files on a DVD-Video are required to be less-than or equal-to 2^{30}-1 bytes.[9]

A UDF-formatted DVD may also include the ISO-9660[6] file-system and so become a UDF-bridge-disc. It appears↳10.4 that all DVD-Videos support the ISO-9660 file-system.

2.8 DVD-Feature

This document uses the term feature to refer to the DVD's user-data; which may hold several video and audio tracks, subtitles and menus. The distinction between a DVD and its feature is relevant for a CSS-encrypted DVD-Video: CSS stores keys in the DVD's lead-in↳2.4 and in the header of the data-frames↳2.6, which is both outside the feature.

The feature area is the UDF file-system↳2.7: a contiguous area of 800_H-byte blocks. The file-system must at least contain a VIDEO_TS.IFO file-record in the VIDEO_TS directory. The first block of the VIDEO_TS.IFO file appears[10] to contain all information and pointers to play the feature. There are additional file-records to describe the content, but they are actually not necessary.[9],↳10.6

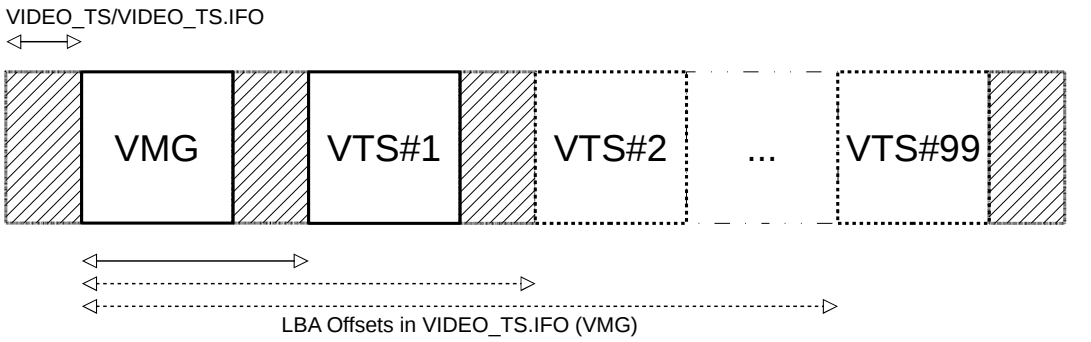


Figure 1: Feature Area

The feature's content[5] starts with the video-manager (VMG) followed by the first video-title-set (VTS) and up to 98 other optional VTS. Both, VMG and VTS, have almost the same structure: First comes the information area (IFO), followed by the optional video-object-stream (VOB), finalized by a backup (BUP) of the initial IFO.

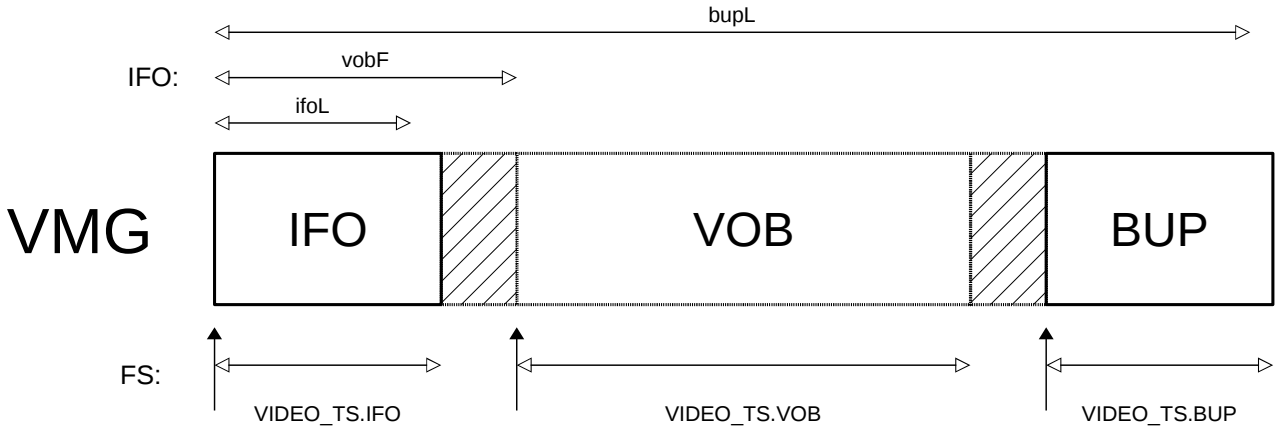


Figure 2: Video Manager (VMG)

The first IFO block[10] holds, besides others, three pointers (each an offset as a number of blocks): a pointer to the last IFO block (ifoL), a pointer to the first VOB block (vobF) which is optional (zero if none) and a pointer to the last BUP block (bupL).

Similar information can be obtained from the records of the file-system. The file records VIDEO_TS.IFO, VIDEO_TS.VOB and VIDEO_TS.BUP hold each the start LBA of the corresponding area, and its size in bytes. The size of each file is a multiple of 800_H. If there is no VOB area, the VIDEO_TS.VOB file-record is omitted, or its file size is set to zero.

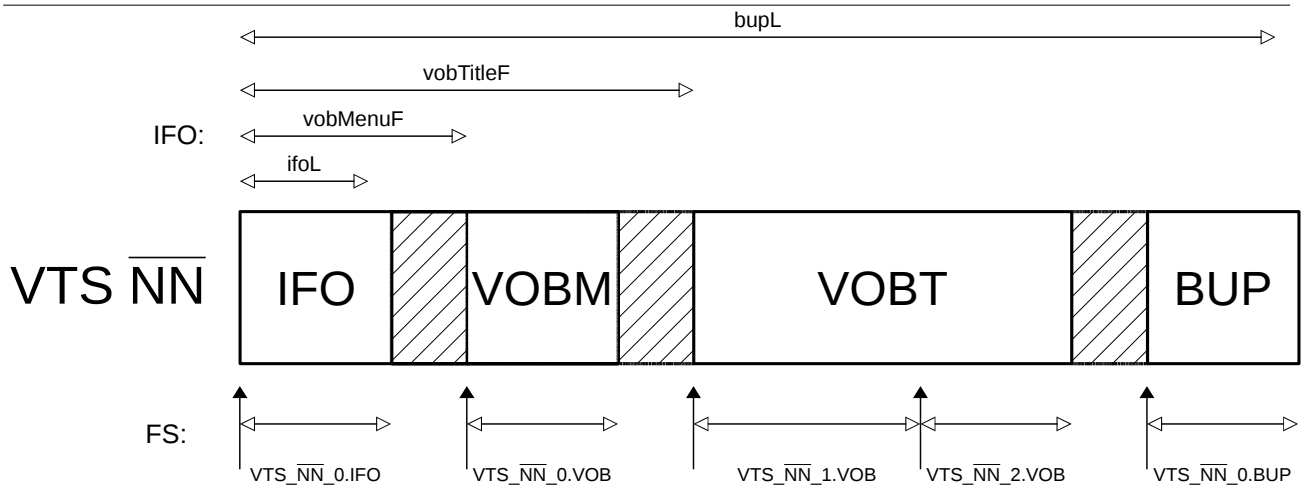


Figure 3: Video Title Set (VTS) with two VOBT files

The structure of the VTS[10] is similar to the VMG. The first IFO block holds, besides others, four pointers (each an offset as a number of blocks): a pointer to the last IFO block (ifoL), a pointer to the first VOB menu block (vobMenuF) which is optional (zero if none), a pointer to the first VOB title block (vobTitleF) and a pointer the last BUP block (bupL).

Similar information can be obtained from the records of the file-system. There are up to 99 VTS, here numbered with two digits: $\overline{NN}$ = 01..99. The files VTS_$\overline{NN}$_0.IFO, VTS_$\overline{NN}$_1.VOB, and VTS_$\overline{NN}$_0.BUP are mandatory. The file VTS_$\overline{NN}$_0.VOB, which contains menu information, is optional. If the VOBT area requires more than 2^{30}-800_H bytes, VTS_$\overline{NN}$_1.VOB is continued by VTS_$\overline{NN}$_2.VOB, and so on, if necessary up to VTS_$\overline{NN}$_9.VOB, each with a maximum size of 2^{30}-800_H bytes. The consecutive VOB files VTS_$\overline{NN}$_1.VOB to VTS_$\overline{NN}$_9.VOB form a contiguous area without gaps. The size of each file is a multiple of 800_H.

2.9 DVD-Pack

A VOB is kind of an MPEG-2 program-stream[11] (PS).[5] A PS consists of so-called Packs. Packs on a DVD-Video are slightly restricted, hence they are named DVD-Pack in the rest of the document. They have a fixed size of 800_H bytes. So each DVD-Pack fits exactly into one logical-block. A DVD-Pack contains one or two packetized elementary streams (PES).

Valid stream identifiers are:[10]

Identifier	Type	Extended
BD_H	Private-1-stream: contains subpictures or non-MPEG-audio	✓
BE_H	Padding-stream: contains FF_H filler bytes	
BF_H	Private-2-stream: contains navigation data	
$C\#_H$	MPEG-audio-stream, the symbol # is used for digits 0 to 7	✓
$E0_H$	Video-stream	✓

A DVD-Pack appears↳10.8 to contain either one or two PES: (BD_H), (BD_H,BE_H), (BE_H), (BF_H,BF_H), ($C\#_H$), ($C\#_H$,BE_H), ($E0_H$) or ($E0_H$,BE_H).

An extended PES holds additional information including a scrambling-control-field↳2.10. This field is used to signal that the DVD-Pack is CSS-encrypted.

2.10 Feature Encryption

If the first PES of a DVD-Pack is a private-1-stream (BD_H), an audio-stream ($C\#_H$) or a video-stream ($E0_H$) then it contains extended information including an MPEG-2-PS 2-bit scrambling-control-field.[11] The field signals that the pack is encrypted if bit 4 or bit 5 (bit-mask 30_H) are set.

The location of the scrambling-control-field in a PS may vary. However, for a DVD-Pack the scrambling-control-field is always located on offset 14_H. Even if the PES is not extended. That is, a DVD-Pack is considered as encrypted if $[14_H] \,\&\, 30_H \neq 00_H$.[4] This value appears↳10.10 to be always zero for navigation data (BF_H).

If a DVD-Pack is marked as encrypted then all data in this DVD-Pack from offset 80_H to $7FF_H$ are encrypted. Data before offset 80_H are never encrypted.[4]

The encryption key is the title-key. Additionally, the five bytes from offset 54_H to 58_H in the DVD-Pack, which are actually ordinary stream data, serve as salt.↳4.5 It appears↳10.19 that one and the same title-key is used for all VMG DVD-Packs. There is also one dedicated title-key for each of the (up to) 99 VTS.

Note: The scrambling-control-field appears↳10.11 to correspond to the CP_MOD field of the CPR_MAI↳2.5 in the frame-header.

3 DVD Drive

The SCSI[7] interface to a DVD drive provides all means to read information from a DVD through multi-media commands (MMC).[8] In SCSI terms, the player-software is the host, the drive is the logical unit (LU).

The MMC read-command is used to retrieve logical-blocks from the user-data area.↳2.1 However, whenever a drive detects a CSS-encrypted DVD-Video↳2.2, it locks all blocks that are marked as copyrighted.↳2.5 An MMC read-command for those blocks fails with an error. To unlock these blocks, an authentication-handshake, that is a sequence of MMC commands, needs to be executed.↳3.1

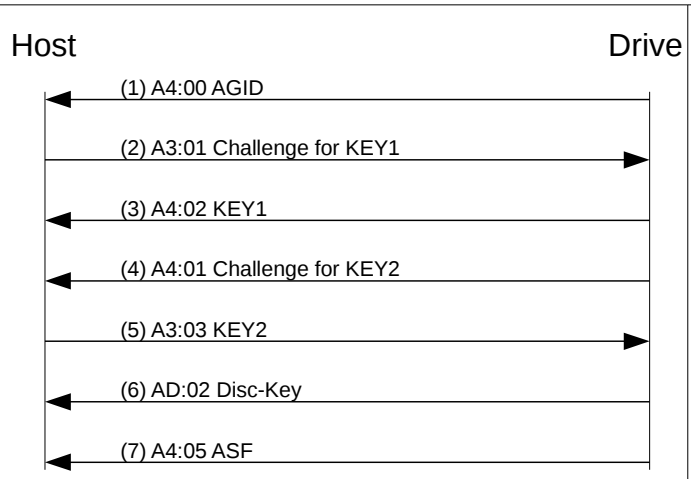
In order to decrypt a DVD-Pack↳2.10, the host has to retrieve the disc-key-block↳2.4 and the title-keys↳2.6. The authentication-handshake provides the disc-key-block. A similar handshake is executed to retrieve title-keys.↳3.2 The disc-key-block and the title-keys are obfuscated↳3.3 by the drive to prevent snooping them on the SCSI bus (even though they are already encrypted).

Also, a licensed drive implements the regional playback control.↳3.4 That is, if the DVD-Video's region-code↳2.3 doesn't match the drive's region-code, the drive is supposed to prevent playback.

7 http://en.wikipedia.org/wiki/SCSI

3.1 Authentication Handshake

The authentication handshake is used to unlock all logical-blocks↳2.1 that are marked as copyrighted↳2.5. Furthermore, the handshake is used retrieve the encrypted disc-key-block↳2.4. The handshake consists of seven MMC commands[8] as illustrated in figure 4.



MMC Commands:

Send Key (operation code $A3_H$)
- key format 01: send challenge for KEY1
- key format 03: send KEY2

Report Key (operation code $A4_H$)
- key format 00: get auth. grant identifier (AGID)
- key format 01: get challenge for KEY2
- key format 02: get KEY1
- key format 05: get authentication success flag (ASF)

Read Disc Structure (operation code AD_H)
- format code 02: get disc-key-block

Figure 4: Authentication Handshake

1. The AGID is a number from 0 to 3. It is allotted by the drive and used as unique session identifier.

2. A 10-byte challenge for KEY1 is sent to the drive. The challenge (also called nonce) is a random number generated by the host. (However, if the host doesn't care, it could always use the same (not-random) value.[4])

3. There are 32 so-called encryption-variants. The drive selects (randomly) one of them, encrypts↳4.6 the challenge for KEY1 and returns the 5-byte result, which is KEY1.

The host encrypts also the challenge for KEY1. It tries all 32 variants until the result matches the returned KEY1. The matching variant is saved by the host for later use.

4. The host requests the 10-byte challenge for KEY2 from the drive. This is a (random) number generated by the drive.

5. The host encrypts the challenge for KEY2 with the saved variant (from step 3). The 5-byte result, which is KEY2, is sent to the drive. The drive verifies KEY2. On mismatch, subsequent commands of the handshake will result in an error.

6. The host requests the encrypted disc-key-block↳2.4. The disc-key-block is obfuscated↳3.3 with the buskey by the drive before transmission. This 5-byte buskey is the encrypted result of KEY1, KEY2 and the saved variant (from step 3). In order to remove the obfuscation, the host has to XOR the disc-key-block with the bus-key. When the drive returns the obfuscated disc-key-block, it also sets the authentication-success-flag (ASF). All previously locked logical-blocks are unlocked, so read access is granted.

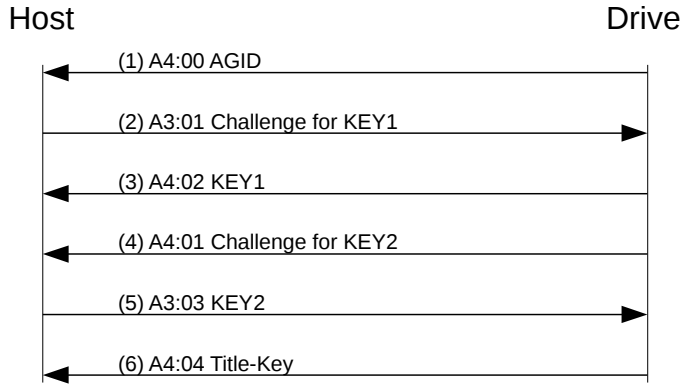
7. Finally the host requests the ASF; and so verifies whether the authentication-handshake was successful.

References: Disc-Key-Block ↳2.4, Copyright Management Information↳2.5, Obfuscation↳3.3, Authentication-Key Encryption↳4.6, AGID Management↳9.2

Note: The drive locks all logical-blocks that are marked as copyrighted; unless authenticated. An MMC read-command (e.g. operation code 28_H or $A8_H$) fails with the error message “scrambled sector without authentication” (sense-code $6F03_H$). However, a read access through the PC's operating system may just report an input-output- (IO-) error, which might be confusing to someone not familiar with CSS.

3.2 Title-Key Handshake

The title-key-handshake is executed to retrieve the encrypted title-keys↳2.6. The handshake consists of six MMC commands[8] as illustrated in figure 5.



MMC Commands:

Send Key (operation code $A3_H$)

- key format 01: send challenge for KEY1
- key format 03: send KEY2

Report Key (operation code $A4_H$)

- key format 00: get auth. grant identifier (AGID)
- key format 01: get challenge for KEY2
- key format 02: get KEY1
- key format 04: get title-key

Figure 5: Title-Key Handshake

The first five commands, which are used for KEY1 / KEY2 exchange, are identical to the authentication-handshake↳3.1. Then, the command to retrieve a logical-block's↳2.1 title-key is issued. The returned 6-byte field contains one byte copyright management information↳2.5 and five bytes with the obfuscated↳3.3 and encrypted↳4.4 title-key. In order to remove the obfuscation from the title-key, the host has to XOR the title-key with the buskey.

References: Copyright Management Information↳2.5, Title-Key↳2.6, Authentication Handshake↳3.1, Obfuscation↳3.3, Title-Key Decryption↳4.4, Authentication-Key Encryption↳4.6

Note: In the context of the MMC[8] specification (section 4.10.2.2), the term *"authentication"* is used for the KEY1 / KEY2 exchange.

3.3 Obfuscation

The disc-key-block and the title-keys, that are transmitted in the handshakes↳3.1,3.2, are obfuscated by the buskey↳4.6. The buskey differs for each session and is a result of KEY1, KEY2 and the encryption-variant.

In order to remove the obfuscation from a disc-key-block *p*, the obfuscated disc-key-block *c* needs to be XORed[8] with the buskey *k*:

$p_i = c_i \oplus k_{i \bmod 5} \mid i=0..7FF_H$, ⊕ is the XOR-operator[9], *mod* is the modulus-operator[10]

In order to remove the obfuscation from a title-key *p*, the obfuscated title-key *c* needs to be XORed with the buskey *k*:

$p_i = c_i \oplus k_i \mid i=0..4,$⊕ is the XOR-operator

3.4 Regional Playback Control

There are RPC-1 and RPC-2 drives.[8] RPC-2 drives appear to implement a rigid region-code↳2.3 management. If the region-code of the RPC-2 drive does not match the region-code of a DVD-Video then the drive can take appropriate measures. It is still possible to change the region of a drive a few times (see sections 6.28.3.1.7 and 6.34.3.1.4[8]). Thereafter the drive is stuck with the last chosen region.

It appears↳9.6 that many RPC-2 drives reject title-key requests↳3.2 on region mismatch. Hence, licensed video playback would not be possible. However, reading user-data and the retrieval of the disc-key-block may still work. Due to the weak CSS design, it is almost always possible to recover the title-keys from the encrypted feature↳6.3, and so to decrypt the feature.

Other RPC-2 drives appear to disable the access to locked blocks↳9.6 or to the complete DVD-Video[11] on region mismatch. On those drives, there is the option to flash an RPC-1 firmware, if available.[12] If this still doesn't work, one can always purchase a drive for the desired region. Only a few regions exist.

8 http://en.wiktionary.org/wiki/XORed

9 http://en.wikipedia.org/wiki/Exclusive_or#Truth_table

10 http://en.wikipedia.org/wiki/Modulo_operation

11 http://www.videolan.org/support/faq.html#DVDS

12 http://www.rpc1.org

4 CSS Cipher

Libdvdcss[4], respectively the publication of the first re-engineered CSS source code by Derek Fawcus[1], are used as basis for the descriptions in this chapter.

There are four different CSS cipher applications as illustrated in figure 6.

(1) decryption of a disc-key with a player-key↳4.3,

(2) decryption of a title-key with a disc-key↳4.4,

(3) decryption of the feature with a title-key↳4.5 and

(4) encryption of a challenge (nonce) and a variant to an authentication-key↳4.6.

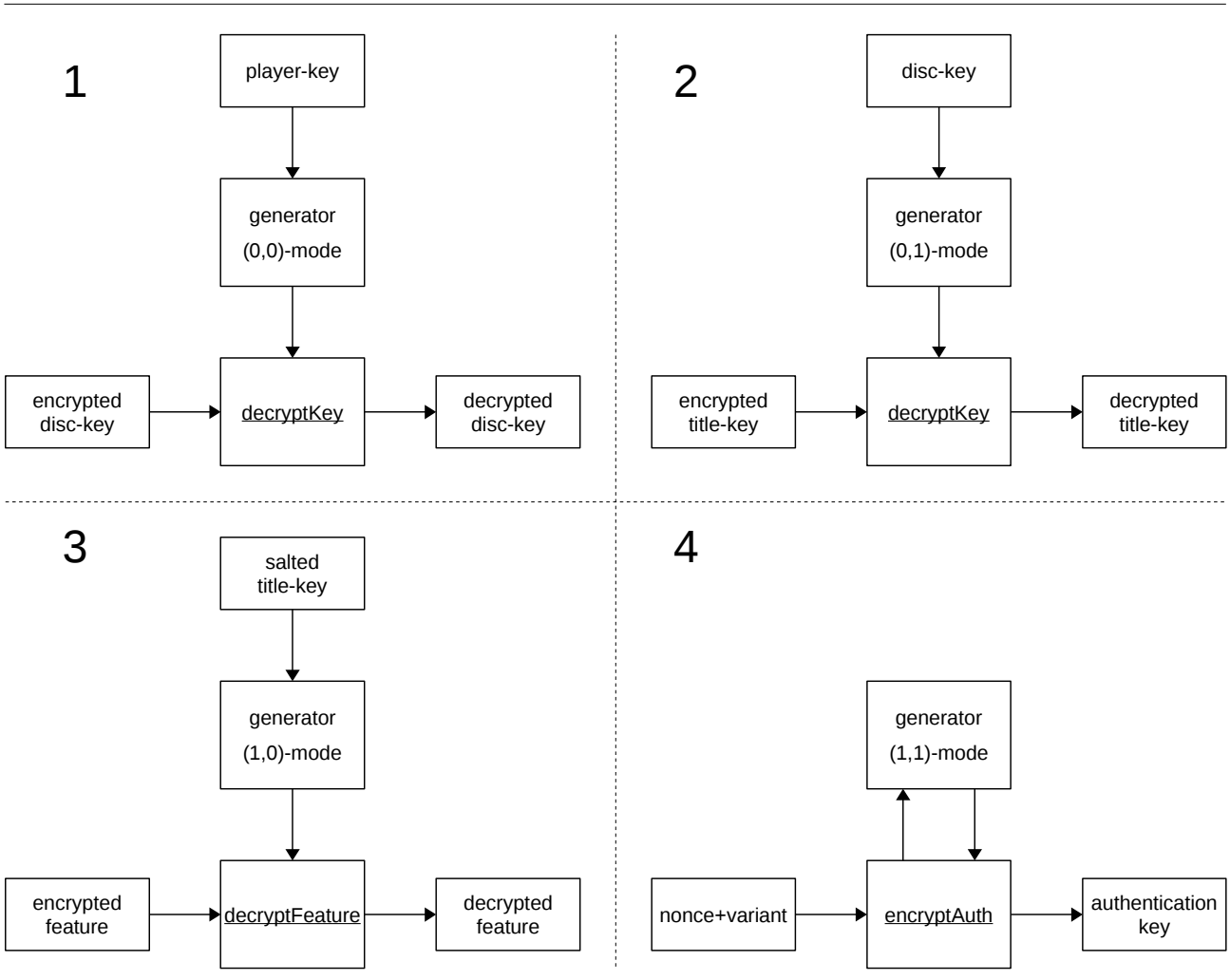


Figure 6: The four CSS Cipher Applications

Three different crypt-functions are involved:

– decryptKey for disc- and title-key decryption↳4.3,

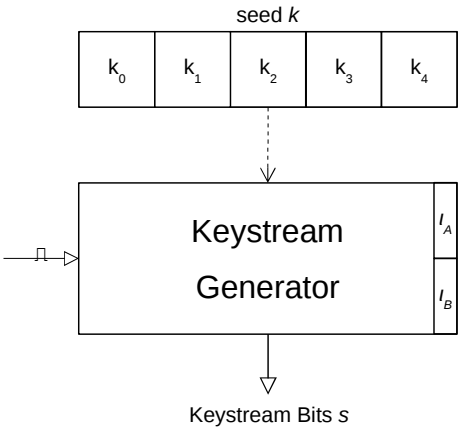
– decryptFeature for feature decryption↳4.5 and

– encryptAuth for authentication-key encryption↳4.6.

All applications use the same keystream generator as basis which is described in the next chapter↳4.1.

4.1 Keystream Generator

The keystream generator[13] is used to produce the keystream[14]. The generator is configured with a 5-byte seed *k* and with two inverting bits I_A and I_B. So it operates in four different (I_A,I_B)-modes for four different kinds of application. With each clock-pulse (⎍) a new keystream bit *s* is produced as illustrated in figure 7.



I_A	I_B	Application
0	0	(1) disc-key decryption
0	1	(2) title-key decryption
1	0	(3) feature decryption
1	1	(4) encryption of authentication keys

Figure 7: Keystream Generator (Overview)

The generator *g* in (I_A,I_B)-mode consists of two LFSR↳4.2 registers, that is the LFSR-A register r_A and the LFSR-B register r_B, and a 1-bit full-adder[15] (ADD) with the carry-over bit *c* as illustrated in figure 8.

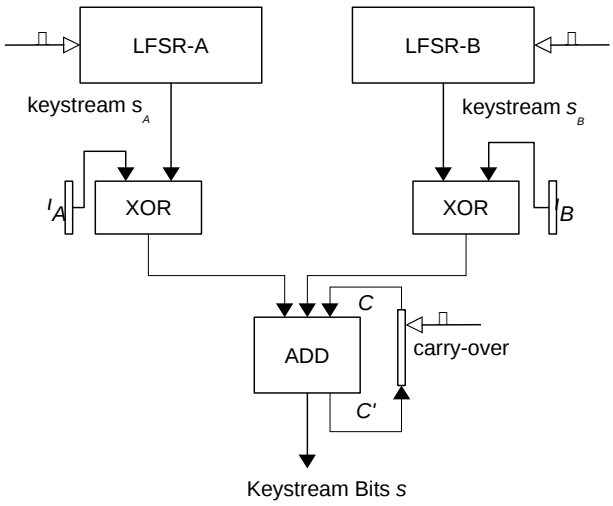


Figure 8: Keystream Generator (Details)

With each clock-pulse (⎍) the generator *g* transitions into its next state *g'* and so a new keystream bit *s* is produced. A generator state *g* can be described as $g = (r_A, r_B, c)$. Initially the first two bytes of the seed *k* are used to set up r_A. The last three bytes of the seed *k* are used to set up r_B.↳4.2 The carry-over bit *c* is set up to 0.

13 http://en.wikipedia.org/wiki/Stream_cipher

14 http://en.wikipedia.org/wiki/Keystream

15 http://en.wikipedia.org/wiki/Adder_%28electronics%29#Full_adder

Each LFSR$^{\hookrightarrow 4.2}$ has a *tap*-method to provide a keystream bit: $s_A := r_A.tap()$ respectively $s_B := r_B.tap()$. The keystream bits s_A and s_B are optionally inverted by XOR[16] (with the bits ι_A and ι_B) and then fed to the 1-bit full-adder (ADD) that provides the generator's keystream bit s and the carry-over bit c':

$$(s, c') := (s_A \oplus \iota_A) + (s_B \oplus \iota_B) + c$$

Each LFSR has a method to transition into the next state: $r_A' := r_A.next()$ respectively $r_B' := r_B.next()$. The method to transition from a generator state g into the next generator state g' can then be described as:

$$g' = g.next() := (r_A', r_B', c')$$

16 http://en.wikipedia.org/wiki/Exclusive_or#Truth_table

4.2 Linear Feedback Shift Register (LFSR)

The keystream generator employs two linear-feedback-shift-registers[17]: LFSR-A and LFSR-B.

LFSR-A consists of a 17-bit register $r = (r_0 ,.., r_{16})$ with $r_i = 0 \mid 1$ for $i = 0..16$.

LFSR-B consists of a 25-bit register $r = (r_0 ,.., r_{24})$ with $r_i = 0 \mid 1$ for $i = 0..24$.

LFSR-A and LFSR-B are set up by a 5-byte seed k as illustrated in figure 9.

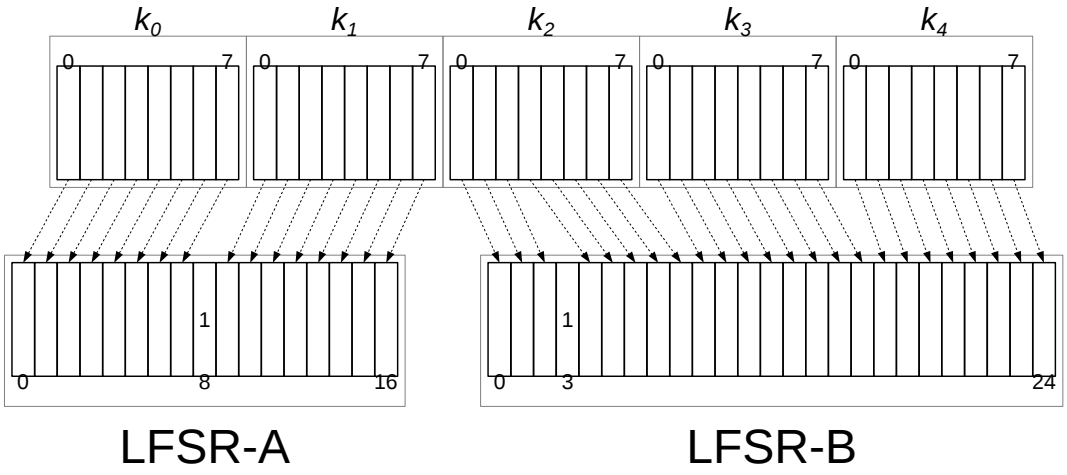


Figure 9: LFSR-A and LFSR-B Setup

LFSR-A is set up with the first two bytes k_0 and k_1 of the seed:

– Bits $(r_0 ,.., r_7)$ are initialized with ($k_{0:0}$,.., $k_{0:7}$), that is $r_i := k_{0:i}$ for $i = 0..7$.

– Bit r_8 is initialized to 1, that is $r_8 := 1$.

– Bits $(r_9 ,.., r_{16})$ are initialized with ($k_{1:0}$,.., $k_{1:7}$), that is $r_{9+i} := k_{1:i}$ for $i=0..7$.

LFSR-B is set up with the last three bytes k_2, k_3 and k_4 of the seed:

– Bits $(r_0 ,.., r_2)$ are initialized with ($k_{2:0}$,.., $k_{2:2}$), that is $r_i := k_{2:i}$ for $i = 0..2$.

– Bit r_3 is initialized to 1, that is $r_3 := 1$.

– Bits $(r_4 ,.., r_8)$ are initialized with ($k_{2:3}$,.., $k_{2:7}$), that is $r_{4+i} := k_{2:3+i}$ for $i=0..4$.

– Bits $(r_9 ,.., r_{16})$ are initialized with ($k_{3:0}$,.., $k_{3:7}$), that is $r_{9+i} := k_{3:i}$ for $i=0..7$.

– Bits $(r_{17} ,.., r_{24})$ are initialized with ($k_{4:0}$,.., $k_{4:7}$), that is $r_{17+i} := k_{4:i}$ for $i=0..7$.

17 http://en.wikipedia.org/wiki/Linear_feedback_shift_register

Both LFSRs are driven through XOR[18] of pre-selected register bits. The bit positions that affect the next LFSR state are called the taps. The tapped value is the keystream bit *s*.

LFSR-A taps bits 2 and 16, that is $s = r.tap() := r_2 \oplus r_{16}$.

LFSR-B taps bits 12, 20, 21 and 24, that is $s = r.tap() := r_{12} \oplus r_{20} \oplus r_{21} \oplus r_{24}$.

To transition (⎍) from an LFSR state *r* into the next LFSR state *r' = r.next()*:

– the tap is applied,

– the *n*-bit register *r* is shifted upwards, that is $r'_{i+1} := r_i \mid i = 0 .. (n-2)$,

– the lowest bit is set to the previously tapped value, that is $r'_0 := r.tap()$.

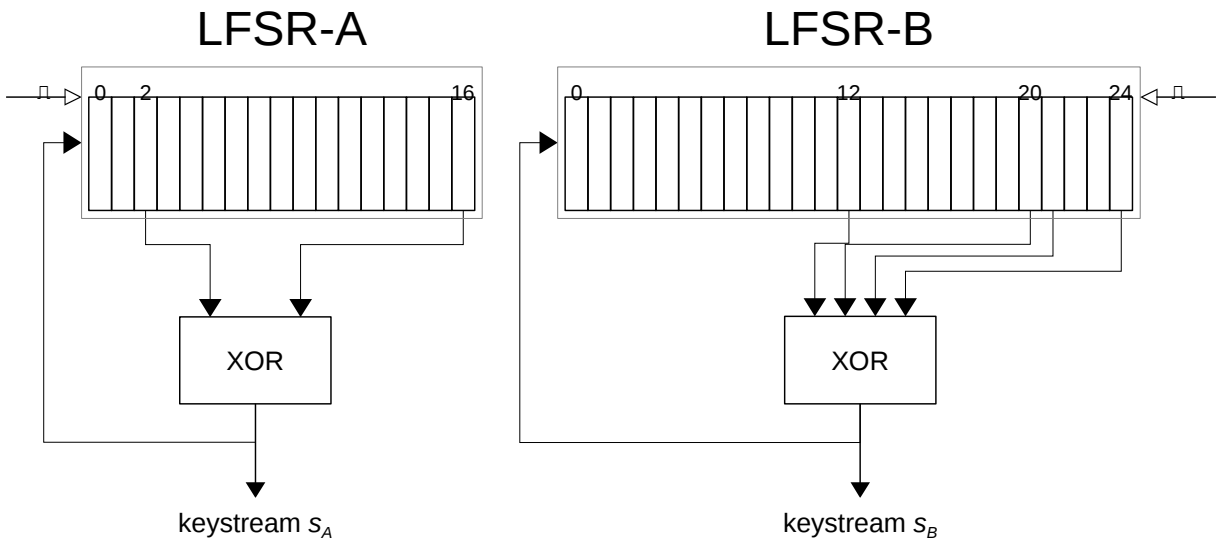


Figure 10: LFSR-A and LFSR-B Transition

If all bits of register *r* are zero, all of *r'* bits are also zero. This zero-register cycle has a period length of 1.

For none-zero registers:

– the LFSR-A cycle has a period with 2^{17}-1 different states↳8.1,

– the LFSR-B cycle has a period with 2^{25}-1 different states↳8.2.

Note: A binary register can be implemented with the least significant bit first (and so the most significant bit last) or with the most significant bit first (and so the least significant bit last). In this document and the accompanied program, the first approach is chosen. Also, illustrations in this document for binary values show the least significant bit left and the most significant bit right, while numbers in the text start with the most significant digit left (as commonly used). For example, the number $A3_H$ is illustrated by its binary representation $1100\text{-}0101_B$.

18 http://en.wikipedia.org/wiki/Exclusive_or#Truth_table

4.3 Disc-Key Decryption

The disc-key[↳2.4] decryption is defined as: *(k , c) → p.*

k: 5-byte player-key

c: 5-byte encrypted disc-key (cipher-text)

p: 5-byte decrypted disc-key (plain-text)

The decryption is a two step operation as illustrated in figure 11. The 5-byte player-key *k* serves as seed for the keystream generator[↳4.1] which operates in (0,0)-mode. The generator produces the 5-byte keystream *s*. The encrypted disc-key *c* and the keystream *s* are fed to the decryptKey-function which returns the decrypted disc-key *p*.

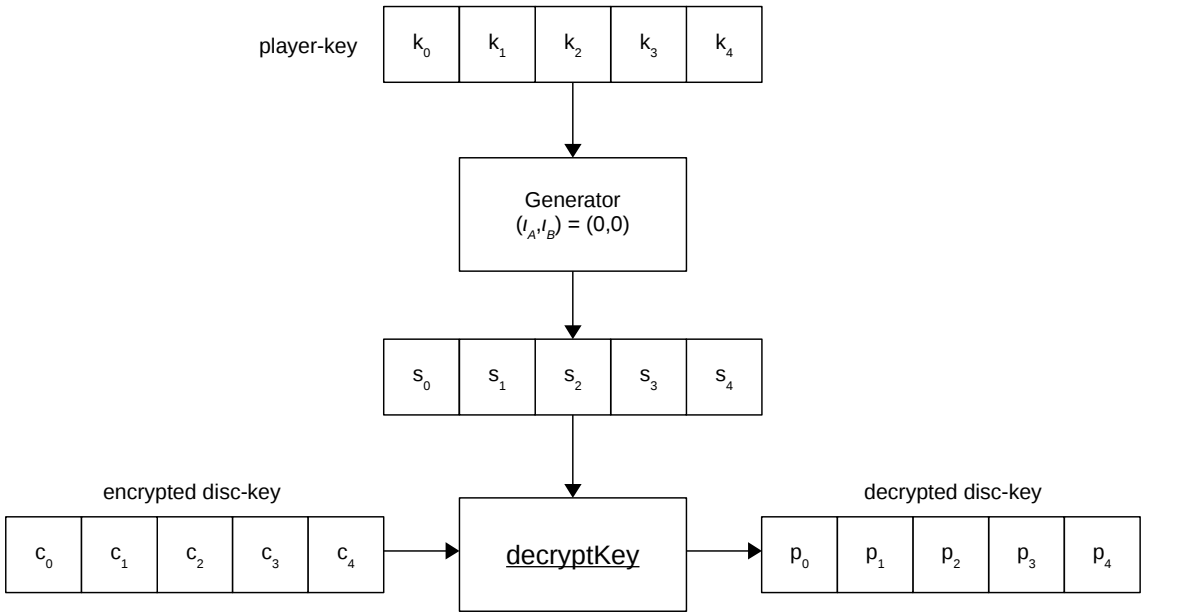


Figure 11: Disc-Key Decryption

The decryptKey-function is a set of 10 equations with two almost identical blocks of five equations. The first block computes an intermediate 5-byte value *t* from the keystream *s* and the encrypted disc-key *c*. The second block computes the decrypted disc-key *p* from the keystream *s* and the intermediate value *t*.

Block # 1		Block # 2	
$t_4 = s_4 \oplus [\ c_4\ ]_{\#A} \oplus c_3$	(eqn. 1)	$p_4 = s_4 \oplus [\ t_4\ ]_{\#A} \oplus t_3$	(eqn. 6)
$t_3 = s_3 \oplus [\ c_3\ ]_{\#A} \oplus c_2$	(eqn. 2)	$p_3 = s_3 \oplus [\ t_3\ ]_{\#A} \oplus t_2$	(eqn. 7)
$t_2 = s_2 \oplus [\ c_2\ ]_{\#A} \oplus c_1$	(eqn. 3)	$p_2 = s_2 \oplus [\ t_2\ ]_{\#A} \oplus t_1$	(eqn. 8)
$t_1 = s_1 \oplus [\ c_1\ ]_{\#A} \oplus c_0$	(eqn. 4)	$p_1 = s_1 \oplus [\ t_1\ ]_{\#A} \oplus t_0$	(eqn. 9)
$t_0 = s_0 \oplus [\ c_0\ ]_{\#A} \oplus t_4$	(eqn. 5)	$p_0 = s_0 \oplus [\ t_0\ ]_{\#A}$	(eqn. 10)

The symbol ⊕ is the XOR-operator. The bracket $[\ ]_{\#A}$ defines a 256-byte lookup-table[↳4.7].

4.4 Title-Key Decryption

The title-key↳2.6 decryption is defined as *(k , c) → p.*

k: 5-byte disc-key

c: 5-byte encrypted title-key (cipher-text)

p: 5-byte decrypted title-key (plain-text)

The decryption is a two step operation as illustrated in figure 12. The 5-byte disc-key *k* serves as seed for the keystream generator↳4.1 which operates in (0,1)-mode. The generator produces the 5-byte keystream *s*. The encrypted title-key *c* and the keystream *s* are fed to the decryptKey-function↳4.3 which returns the decrypted title-key *p*.

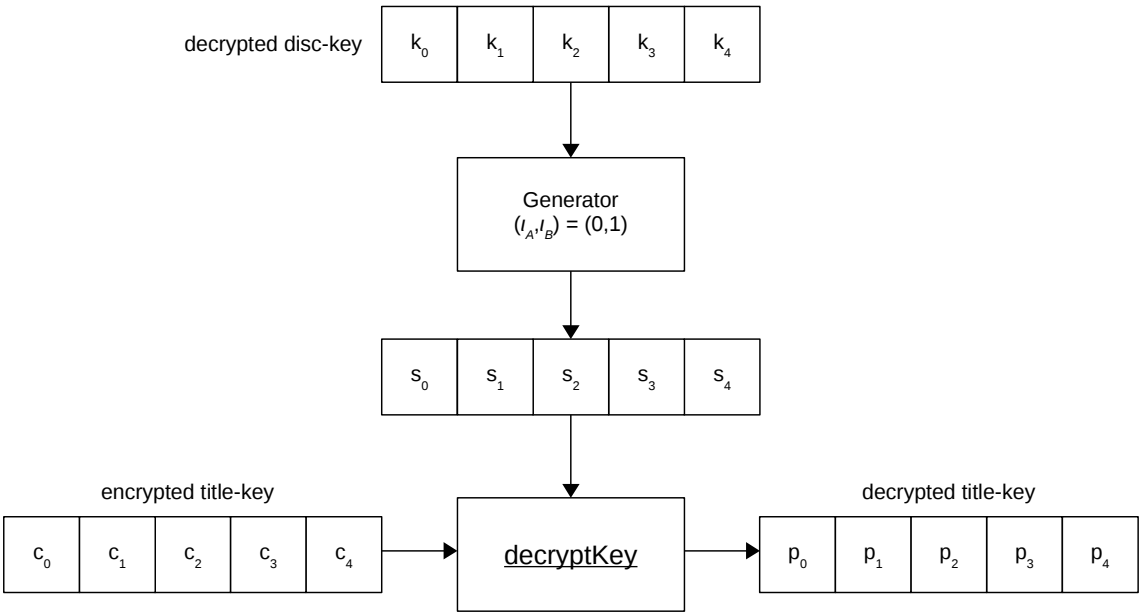


Figure 12: Title-Key Decryption

Note: The title-key decryption is almost identical to the disc-key decryption↳4.3. Only the generator operates in (0,1)-mode instead of (0,0)-mode.

4.5 Feature Decryption

The decryption of a DVD-Pack↳2.10 is defined as: *(k , z , c) → p.*

k: 5-byte title-key

z: 5-byte salt

c: 780_H-byte encrypted feature (cipher-text)

p: 780_H-byte decrypted feature (plain-text)

The decryption is a three step operation as illustrated in figure 13. The title-key *k* and the salt *z* are XORed. The result serves as seed for the keystream generator↳4.1 which operates in (1,0)-mode. The generator produces a 780_H-byte keystream *s*. The encrypted feature *c* and the keystream *s* are fed to the decryptFeature-function which returns the decrypted feature *p*.

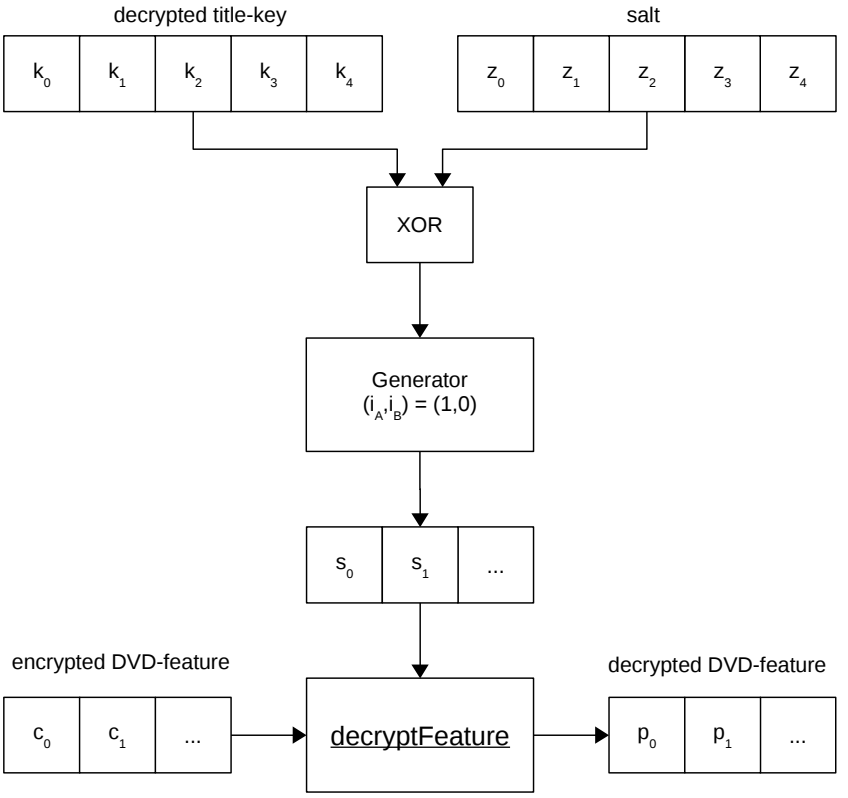


Figure 13: Feature Decryption

The decryptFeature-function is defined as follows: $p_i := s_i \oplus [\ c_i\ ]_{\#A}$ where $i=0..77F_H$.

The symbol ⊕ is the XOR-operator. The bracket $[\ ]_{\#A}$ defines a 256-byte lookup-table↳4.7.

4.6 Authentication-Key Encryption

The encryption of an authentication key is defined as: *(p , v) → c.*

p: 10-byte challenge (nonce[19]) (plain-text)

v: 5-bit variant number 0..31

c: 5-byte authentication key (cipher-text)

The generator operates in (1,1)-mode and produces a 30-byte keystream *s*. The 5-byte seed *k* for the generator is extracted from the nonce. This is illustrated in figure 14, and in more detail in figure 15.

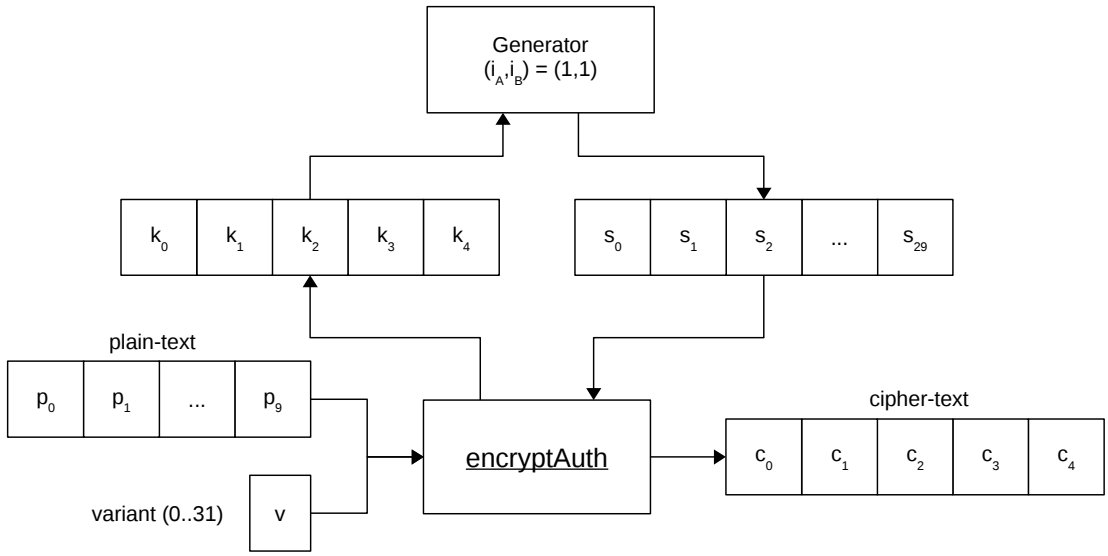


Figure 14: Overview: Encryption of Authentication-Keys

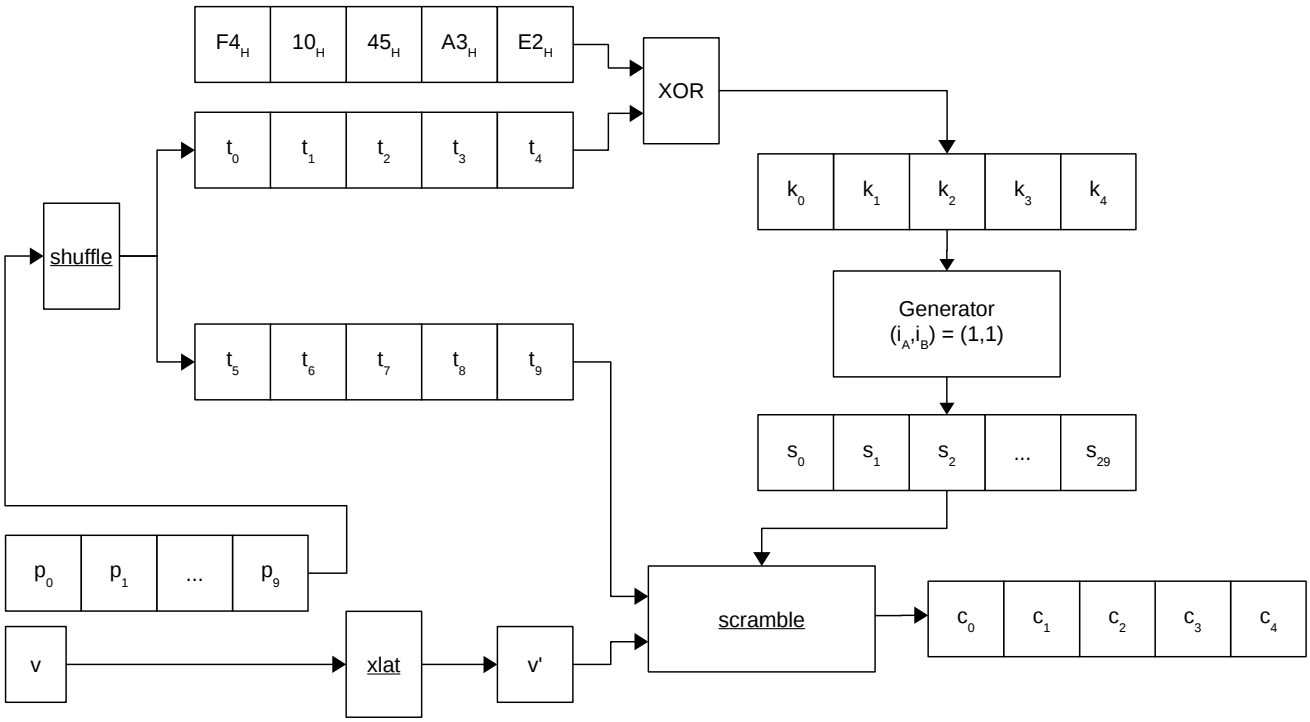


Figure 15: Details: Encryption of Authentication-Keys

19 http://en.wikipedia.org/wiki/Cryptographic_nonce

The shuffle-function splits up and transforms the 10-byte nonce p into two 5-byte blocks. The first block (t_0 ,.., t_4) is XORed with the 5-byte sequence ($F4_H$,10_H,45_H,$A3_H$,$E2_H$). The result is the 5-byte seed k for the keystream generator. The second block (t_5 ,.., t_9) serves as one of the three inputs to the scramble-function. The variant v is translated by the xlat-function to the 1-byte variant v', which serves as another input to the scramble-function. Finally, the 30-byte keystream s serves as third input to the scramble-function. The result of the scramble-function is the 5-byte authentication key c.

So, there are three functions: scramble, shuffle and xlat.

The scramble-function $(v' , t_5 ,.., t_9 , s) \rightarrow c$ consists of the following 14 equations:

1:	$c = (t_5 ,.., t_9)$				
2:	$c = f (s_0 ,.., s_4, v', c)$			3:	$c_0 = c_0 \oplus c_4$
4:	$c = f (s_5 ,.., s_9, v', c)$			5:	$c_0 = c_0 \oplus c_4$
6:	$c = f (s_{10} ,.., s_{14}, v', c)$	7:	$c_1 = [c_1]_{\#D}$	8:	$c_0 = c_0 \oplus c_4$
9:	$c = f (s_{15} ,.., s_{19}, v', c)$	10:	$c_1 = [c_1]_{\#D}$	11:	$c_0 = c_0 \oplus c_4$
12:	$c = f (s_{20} ,.., s_{24}, v', c)$			13:	$c_0 = c_0 \oplus c_4$
14:	$c = f (s_{25} ,.., s_{29}, v', c)$				

The symbol ⊕ is the XOR-operator.

The function $c = f (s_{n+0} ,.., s_{n+4}, v', c)$ is defined as:

$$c_0 = [[c_0 \oplus s_{n+0}]_{\#B} \oplus v']_{\#C}$$
$$c_i = [[c_i \oplus s_{n+i}]_{\#B} \oplus v']_{\#C} \oplus c_{i-1} \mid i = 1..4$$

Brackets $[ ]_{\#B}$, $[ ]_{\#C}$, and $[ ]_{\#D}$ are used for lookup-tables↳4.7.

The shuffle-function $(p_0 ,.., p_9) \rightarrow (t_0 ,.., t_9)$ describes a permutation $(\sigma(p_0) \cdots \sigma(p_9))$[20]. That is, $(p_0 ,.., p_9)$ and $(t_0 ,.., t_9)$ contain exactly the same ten elements, however, the order may be different.

The xlat-function $v \rightarrow v'$ is a 32-byte lookup-table. That is, $v' := [v]$, which is the value at address v in the lookup-table. It transforms the variant v, which is in the range from 0 to 31, to the 1-byte value v'.

For the authentication handshake between host and drive↳3.1,3.2 three types of encryption are used: the encryption of KEY1, the encryption of KEY2 and the encryption of the buskey. In each of these cases there are 10 bytes of plain-text p (e.g. a nonce) and a 5-bit variant v in the range from 0 to 31.

For the KEY1-encryption, the 10-byte challenge, that is sent by the host to the drive, serves as plain-text p. The drive encrypts p with a randomly chosen variant v. The cipher-text c, which is KEY1, is sent back to the host. So the host knows p and c, but not v, yet. To determine the value of v, the host performs the encryption of p for each possible $v=0..31$, until the cipher-text c matches the received KEY1.

For the KEY2-encryption, the 10-byte challenge, that is sent by the drive to the host, serves as plain-text p. Both sides encrypt p and the variant v, that is the same variant v that was chosen for KEY1, to the cipher-text c, which is KEY2.

20 http://en.wikipedia.org/wiki/Permutation

For the buskey-encryption, the concatenation of KEY1 and KEY2 serves a plain-text *p*. That is $p = KEY1_0 ,.., KEY1_4 , KEY2_0 ,.., KEY2_4$. Both sides encrypt *p* and the variant *v*, that is the same variant *v* that was chosen for KEY1 and KEY2 encryption, to the cipher-text *c*, which is the buskey.

For each of these three encryption types, different permutation- and lookup-tables are used.

	σ: p → t	Lookup-Table *[ ]* for v → *v'* (hexadecimal)
KEY1	(1 5 3 0 7 4 2 9 6 8)	[00 01 04 05 10 11 14 15 20 21 24 25 30 31 34 35 80 81 84 85 90 91 94 95 A0 A1 A4 A5 B0 B1 B4 B5]
KEY2	(7 9 5 2 4 1 6 0 8 3)	[24 20 34 30 25 21 35 31 A4 A0 B4 B0 A5 A1 B5 B1 04 00 14 10 05 01 15 11 84 80 94 90 85 81 95 91]
Buskey	(5 3 8 6 2 7 9 1 4 0)	[84 A4 94 B4 04 24 14 34 80 A0 90 B0 00 20 10 30 85 A5 95 B5 05 25 15 35 81 A1 91 B1 01 21 11 31]

Figure 16: Authentication: shuffle- and xlat-values

For example, *p → t* for KEY1: $(p_1, p_5, p_3, p_0, p_7, p_4, p_2, p_9, p_6, p_8)$

For example, v → *v'* for KEY2: $v'(0) = 24_H$, $v'(1) = 20_H$, $v'(2) = 34_H$, ..., $v'(31) = 91_H$

4.7 Lookup Tables[21]

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
00	33	73	3B	26	63	23	6B	76	3E	7E	36	2B	6E	2E	66	7B	D3	93	DB	06	43	03	4B	96	DE	9E	D6	0B	4E	0E	46	9B
20	57	17	5F	82	C7	87	CF	12	5A	1A	52	8F	CA	8A	C2	1F	D9	99	D1	00	49	09	41	90	D8	98	D0	01	48	08	40	91
40	3D	7D	35	24	6D	2D	65	74	3C	7C	34	25	6C	2C	64	75	DD	9D	D5	04	4D	0D	45	94	DC	9C	D4	05	4C	0C	44	95
60	59	19	51	80	C9	89	C1	10	58	18	50	81	C8	88	C0	11	D7	97	DF	02	47	07	4F	92	DA	9A	D2	0F	4A	0A	42	9F
80	53	13	5B	86	C3	83	CB	16	5E	1E	56	8B	CE	8E	C6	1B	B3	F3	BB	A6	E3	A3	EB	F6	BE	FE	B6	AB	EE	AE	E6	FB
A0	37	77	3F	22	67	27	6F	72	3A	7A	32	2F	6A	2A	62	7F	B9	F9	B1	A0	E9	A9	E1	F0	B8	F8	B0	A1	E8	A8	E0	F1
C0	5D	1D	55	84	CD	8D	C5	14	5C	1C	54	85	CC	8C	C4	15	BD	FD	B5	A4	ED	AD	E5	F4	BC	FC	B4	A5	EC	AC	E4	F5
E0	39	79	31	20	69	29	61	70	38	78	30	21	68	28	60	71	B7	F7	BF	A2	E7	A7	EF	F2	BA	FA	B2	AF	EA	AA	E2	FF

Figure 17: Lookup-Table [ ]$_{\#A}$ (Application 1,2 and 3)

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
00	C4	CD	CE	CB	C8	C9	CA	CF	CC	C5	C6	C3	C0	C1	C2	C7	14	1D	1E	1B	18	19	1A	1F	1C	15	16	13	10	11	12	17
20	24	2D	2E	2B	28	29	2A	2F	2C	25	26	23	20	21	22	27	34	3D	3E	3B	38	39	3A	3F	3C	35	36	33	30	31	32	37
40	04	0D	0E	0B	08	09	0A	0F	0C	05	06	03	00	01	02	07	D4	DD	DE	DB	D8	D9	DA	DF	DC	D5	D6	D3	D0	D1	D2	D7
60	E4	ED	EE	EB	E8	E9	EA	EF	EC	E5	E6	E3	E0	E1	E2	E7	F4	FD	FE	FB	F8	F9	FA	FF	FC	F5	F6	F3	F0	F1	F2	F7
80	44	4D	4E	4B	48	49	4A	4F	4C	45	46	43	40	41	42	47	94	9D	9E	9B	98	99	9A	9F	9C	95	96	93	90	91	92	97
A0	A4	AD	AE	AB	A8	A9	AA	AF	AC	A5	A6	A3	A0	A1	A2	A7	B4	BD	BE	BB	B8	B9	BA	BF	BC	B5	B6	B3	B0	B1	B2	B7
C0	84	8D	8E	8B	88	89	8A	8F	8C	85	86	83	80	81	82	87	54	5D	5E	5B	58	59	5A	5F	5C	55	56	53	50	51	52	57
E0	64	6D	6E	6B	68	69	6A	6F	6C	65	66	63	60	61	62	67	74	7D	7E	7B	78	79	7A	7F	7C	75	76	73	70	71	72	77

Figure 18: Lookup-Table [ ]$_{\#B}$ (Authentication)

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
00	C4	24	14	34	CE	2E	1E	3E	CD	2D	1D	3D	CB	2B	1B	3B	44	A4	94	B4	4E	AE	9E	BE	4D	AD	9D	BD	4B	AB	9B	BB
20	04	E4	D4	F4	0E	EE	DE	FE	0D	ED	DD	FD	0B	EB	DB	FB	84	64	54	74	8E	6E	5E	7E	8D	6D	5D	7D	8B	6B	5B	7B
40	CC	2C	1C	3C	C6	26	16	36	C5	25	15	35	C3	23	13	33	4C	AC	9C	BC	46	A6	96	B6	45	A5	95	B5	43	A3	93	B3
60	0C	EC	DC	FC	06	E6	D6	F6	05	E5	D5	F5	03	E3	D3	F3	8C	6C	5C	7C	86	66	56	76	85	65	55	75	83	63	53	73
80	C8	28	18	38	CA	2A	1A	3A	C9	29	19	39	CF	2F	1F	3F	48	A8	98	B8	4A	AA	9A	BA	49	A9	99	B9	4F	AF	9F	BF
A0	08	E8	D8	F8	0A	EA	DA	FA	09	E9	D9	F9	0F	EF	DF	FF	88	68	58	78	8A	6A	5A	7A	89	69	59	79	8F	6F	5F	7F
C0	C0	20	10	30	C2	22	12	32	C1	21	11	31	C7	27	17	37	40	A0	90	B0	42	A2	92	B2	41	A1	91	B1	47	A7	97	B7
E0	00	E0	D0	F0	02	E2	D2	F2	01	E1	D1	F1	07	E7	D7	F7	80	60	50	70	82	62	52	72	81	61	51	71	87	67	57	77

Figure 19: Lookup-Table [ ]$_{\#C}$ (Authentication)

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
00	00	81	03	82	06	87	05	84	0C	8D	0F	8E	0A	8B	09	88	18	99	1B	9A	1E	9F	1D	9C	14	95	17	96	12	93	11	90
20	30	B1	33	B2	36	B7	35	B4	3C	BD	3F	BE	3A	BB	39	B8	28	A9	2B	AA	2E	AF	2D	AC	24	A5	27	A6	22	A3	21	A0
40	60	E1	63	E2	66	E7	65	E4	6C	ED	6F	EE	6A	EB	69	E8	78	F9	7B	FA	7E	FF	7D	FC	74	F5	77	F6	72	F3	71	F0
60	50	D1	53	D2	56	D7	55	D4	5C	DD	5F	DE	5A	DB	59	D8	48	C9	4B	CA	4E	CF	4D	CC	44	C5	47	C6	42	C3	41	C0
80	C0	41	C3	42	C6	47	C5	44	CC	4D	CF	4E	CA	4B	C9	48	D8	59	DB	5A	DE	5F	DD	5C	D4	55	D7	56	D2	53	D1	50
A0	F0	71	F3	72	F6	77	F5	74	FC	7D	FF	7E	FA	7B	F9	78	E8	69	EB	6A	EE	6F	ED	6C	E4	65	E7	66	E2	63	E1	60
C0	A0	21	A3	22	A6	27	A5	24	AC	2D	AF	2E	AA	2B	A9	28	B8	39	BB	3A	BE	3F	BD	3C	B4	35	B7	36	B2	33	B1	30
E0	90	11	93	12	96	17	95	14	9C	1D	9F	1E	9A	1B	99	18	88	09	8B	0A	8E	0F	8D	0C	84	05	87	06	82	03	81	00

Figure 20: Lookup-Table [ ]$_{\#D}$ (Authentication)

21 http://en.wikipedia.org/wiki/Lookup_table

5 DVD-Video Decryption

The illustration below describes how to decrypt the feature[↳2.8] of a CSS-encrypted DVD-Video.

I	Verify whether the DVD is CSS-encrypted[↳2.2]		
II	Execute the authentication-handshake[↳3.1] to unlock read-protected logical-blocks[↳2.5]. The handshake is also used to retrieve the encrypted disc-key-block[↳2.4].		
III	Determine the disc-key by decrypting the disc-key-block with a player-key.[↳2.4]		
IV	For each: the feature's video manager (VMG) and all video title sets (VTS)[↳2.8]:		
	A	Choose any logical block address (LBA) of the VMG / VTS VOB area (for example, the first LBA of the VOB) and retrieve the encrypted title-key for this LBA by execution of the title-key-handshake[↳3.2].	
	B	Decrypt the encrypted title-key with the disc-key.[↳4.4]	
	C	For all DVD-Packs:	
		(1)	If the DVD-Pack is marked as encrypted[↳2.10] then decrypt the DVD-Pack with the title-key[↳4.5].

Note:

It is possible to access all read-protected logical-blocks without using a re-engineered authentication handshake (see next paragraph). This might come in handy for backup purposes in countries where backups are legal, but breaking a so-called "active copy protection" is illegal.[22]

A licensed DVD player-software always executes the authentication handshake before the playback starts. This unlocks the read-protection, system-wide! So all it needs to read (and save) the complete feature is to start a licensed player-software.

A licensed player-software comes mostly with each new DVD drive. This makes the CSS mechanism for read-protection virtually useless.

However, this approach does not provide the disc- and title-keys to decrypt the feature. To retrieve them, the re-engineered handshakes need to be executed. Though, it appears to be almost always possible to play an encrypted feature without the original disc- and title-keys by recovering the title-keys from the encrypted feature.[↳6.3]

22 http://de.wikipedia.org/wiki/Privatkopie#Technische_Schutzma.C3.9Fnahmen

6 Cryptanalysis

CSS employs cryptographic keys with a size of only 40 bits. This makes CSS vulnerable to a brute-force attack[23]. That is, a disc-key, a player-key or a title-key can be guessed in a matter of hours on an entry-level computer by testing all possible 2^{40} keys. Even worse, due to the weak CSS design, there are exploits to guess a key within a few seconds or even within a fraction of a second.

(1) In 1999, Derek Fawcus[1] published an implementation for the drive's handshakes↳3.1,3.2 and for the CSS ciphers↳4. Although the program wasn't usable without one of the secret player-keys, it was invaluable because it revealed the details of the CSS design.

(2) On 27 Oct 1999, Frank A. Stevenson[2] published the exploit to recover the seed↳6.6 from a generator's keystream at a complexity of 2^{16}. One day later, on 28 Oct 1999, he published the exploit to recover the keystream↳6.5 from an encrypted disc-key at a complexity of 2^{8}.

As a disc-key-block↳2.4 holds 408 player-key-encrypted disc-keys, it became now possible to recover all the player-keys that were used for this disc-key-block within seconds. Since a player-key is kind of a master key, the CSS encryption was virtually broken at this point. Still, it required a single decrypted disc-key to recover all the player-keys. And it took probably a few days to recover a disc-key by brute force on a consumer PC at this time.

On 30 Oct 1999, Frank A. Stevenson published the exploit to recover the disc-key↳6.4 from its hash-value↳2.4 at a complexity of 2^{25} in less than 20 seconds. Now it was possible to reveal all the secret player-keys within less than a minute on a consumer PC. The CSS design was not only proven to be flawed, it couldn't be taken seriously at all.

(3) There was still a little issue with the regional playback control.↳3.4 If the DVD-Video doesn't match the drive's region, access to title-keys won't be granted. However the (encrypted) feature may still be readable. Ethan Hawke[3] published in 2000 an approach to predict↳6.9 plain-text feature data. This kind of prediction can be used to recover title-keys. It needs only one correct prediction in all of the encrypted DVD-Packs↳2.10 to decrypt a complete VMG/VTS.

(4) The encryption of authentication keys↳3.1,3.2 was already completely re-engineered↳4.6. It's the same for all drives and it's merely a security-through-obscurity[24] mechanism. No keys are involved. Hence, further analysis appears to be pointless.

23 http://en.wikipedia.org/wiki/Brute-force_attack

24 http://en.wikipedia.org/wiki/Security_through_obscurity

6.1 Disc-Key

Each DVD-Video holds a disc-key-block↳2.4. The first entry is the hash-value. If the genuine disc-key *k* is decrypted with the hash-value *c* it will result in the disc-key itself: $(k , c) \rightarrow k$.

It is feasible to decrypt all possible 2^{40} disc-keys with the hash-value. This produces a set of matching disc-keys at a complexity of 2^{40} within a matter of hours.↳8.3 Even better, an exploit↳6.4 of the weak CSS design reduces the complexity to 2^{25} and provides the set of matching disc-keys within a few seconds.↳8.4

The disc-key recovery, regardless whether by brute-force or exploit, may result in several matching disc-keys. There might be also no disc-key at all if the hash-value is invalid: for example, there is no disc-key for the hash-value $(00_H,00_H,00_H,00_H,02_H)$.↳8.3,8.4

If more than one disc-key is found, Frank A. Stevenson[2] suggests the decryption of the feature with each of these keys. The genuine disc-key is then selected by actually watching the feature, or, by verifying the integrity of the feature's content.

An alternative approach, without walking into the details of the feature's content, is using the disc-key-block to recover all player-keys by exploit.↳6.2 The disc-key which produces most player-keys appears to be the genuine disc-key.↳10.16

6.2 Player-Keys

Each DVD contains a disc-key-block with 408 player-key-encrypted disc-key variants.[↳2.4] That is, there are *i=1..408* variants c_i of the disc-key *p*, each of them encrypted with a player-key k_i: (k_i , c_i) → *p* [↳4.3].

If the disc-key *p* is known then it will be feasible to test all possible 2^{40} player-keys k_i for any variant c_i. This provides a set of matching player-keys at a complexity of 2^{40} for the variant c_i within a matter of hours.[↳8.5] Even better, an exploit of the weak CSS design reduces the complexity to $\leq 2^{24}$ and provides a set of matching player-keys within a fraction of a second.[↳8.6]

The exploit is a 4-step operation:
[1] Recover all $\leq 2^8$ possible 40-bit keystreams ($\overline{\text{decryptKey}}$) for the variant *c* and disc-key *p*.[↳6.5]
[2] Recover all 2^{16} generators, one for each of the 25-bit keystream heads (s_0 ,.., s_{24}).[↳6.6]
[3] Single out those generators that match the 15-bit keystream tails (s_{25} ,.., s_{39}).[↳6.7]
[4] Cycle back[↳6.8] the generators to the start state and extract the seed, which is a matching player-key.

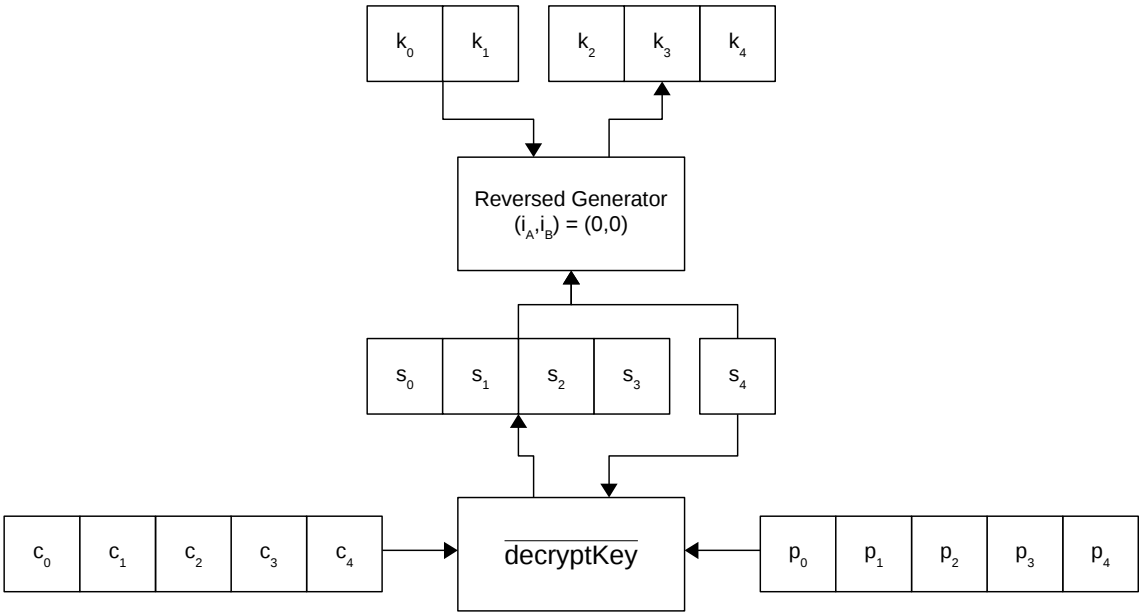


Figure 21: Player-Key Recovery by Exploit

The procedure to recover a player-key, either by brute force or through exploit, may result in none, one or several matching player-keys.[↳10.15]

If the $\overline{\text{decryptKey}}$-function provides no keystream then there is no player-key.

If the $\overline{\text{decryptKey}}$-function provides one keystream then there is one set of player-keys. The set may contain none, one or several player-keys.[↳8.8] If there are several player-keys, they are all equivalent (since they produce the same keystream).

If the $\overline{\text{decryptKey}}$-function provides several keystream then there are several sets of player-keys. The player-keys within a set (if there are several) are all equivalent (since they produce the same keystream).

In case there are several sets of matching player-keys, the genuine set of player-keys has to be sorted out. This can be done by analyzing various encrypted DVD-Videos. It appears ↳10.17 there are 32 genuine sets of player-keys.

Note: Frank A. Stevenson[2] writes in his paper: *"Selecting the correct start state is not a problem, as this attack is used in situations where only the first five output bytes are of significance".* However, this is only true if there is just one matching keystream. In his e-mail (28 Oct 1999) he writes correctly: *"[collisions are] a concern when searching for player-keys, as they have to be eliminated by checking against other discs"*.

6.3 Title-Keys

The decryption of the feature is defined as: $(k , z , c) \rightarrow p$ ↳4.5. In order to recover a title-key *k*, it is necessary to predict a plain-text snippet of the feature *c*. With such a snippet, the keystream correlation exploit↳6.6 enables the recovery of a title-key within a fraction of a second.↳10.22 Though, feature prediction may be expensive or may even fail.

The exploit is a 4-step operation:

[1] Predict a plain-text feature snippet.↳6.9

[2] Derive the keystream *s* from the predicted plain-text DVD-Pack snippet *p* and the corresponding encrypted DVD-Pack snippet *c*: $s_i := p_i \oplus [c_i]$ ↳4.5

[3] Single out the generator's start state for the keystream *s*.↳ 6.6,6.7,6.8

[4] XOR the start state's seed with the salt. The result is a matching title-key.↳2.10

Notes:

– There are mainly two reasons for a missing title-key. First, a drive may deny access to the encrypted title-keys if the region-codes of DVD-Video and drive do not match.↳3.4 Second, only a copy (backup) of the encrypted feature is available, which includes neither a disc-key nor title-keys.↳5

– Depending on the quality of the feature prediction, even a brute-force attack might be feasible.

6.4 Disc-Key Exploit

The exploit to recover a disc-key *k* at a complexity of 2^{25} from its hash-value↳2.4 *c* was presented by Frank A. Stevenson.[2]

Step 1: For all 2^{24} 3-byte tuples (k_0, t_0, t_1) the values $(k_1, s_0, s_1, s_4, t_4)$ are calculated. This is done by transformation of the equations of the decryptKey-function↳4.3.

$s_0 = k_0 \oplus [t_0]$ (eqn. 10)
$t_4 = s_0 \oplus [c_0] \oplus t_0$ (eqn. 5)
$s_4 = t_4 \oplus [c_4] \oplus c_3$ (eqn. 1)
$s_1 = t_1 \oplus [c_1] \oplus c_0$ (eqn. 4)
$k_1 = s_1 \oplus [t_1] \oplus t_0$ (eqn. 9)

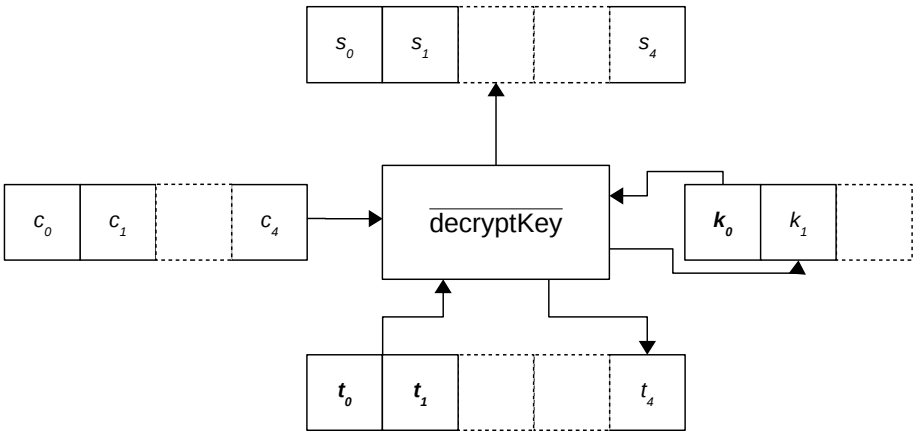


Figure 22: Disc-Key Exploit: Step 1

Step 2.1

$(s_{A_0}, s_{A_1}, s_{A_4}) = LFSR\text{-}A (k_0, k_1)$

Step 2.2

$(s_{B_0}, s_{B_1}) = (s_0, s_1) - (s_{A_0}, s_{A_1})$

$s_{B_{4,0}} = s_4 - s_{A_4}$

$s_{B_{4,1}} = s_4 - s_{A_4} - 1$

Step 2.3

$(k_2, k_3, k_4) = [(s_{B_0}, s_{B_1}, s_{B_4})]_{LFSR\text{-}B}$

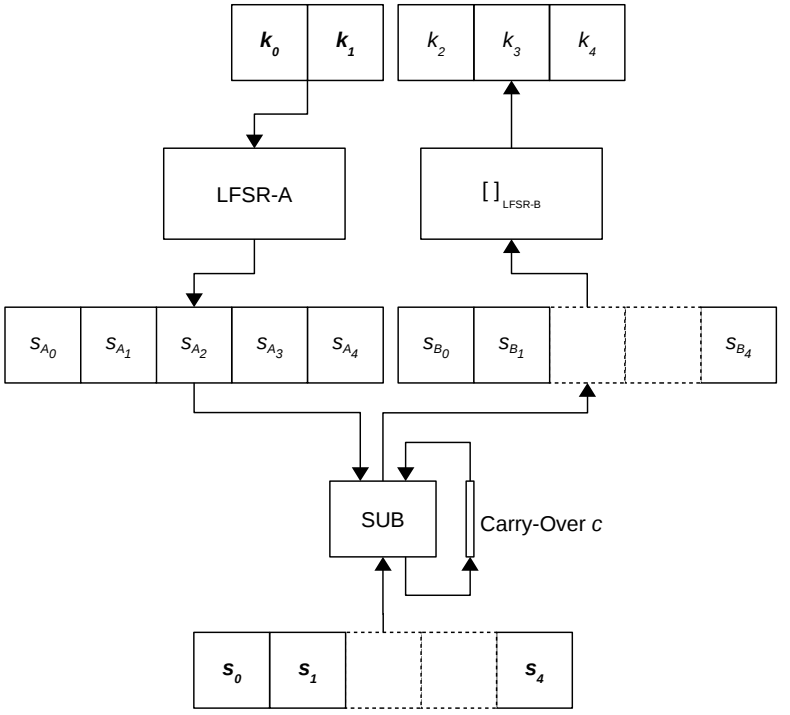


Figure 23: Disc-Key Exploit: Step 2

Step 2: Deduce (k_2, k_3, k_4) from (k_0, k_1) and (s_0, s_1, s_4):

Step 2.1: (k_0, k_1) are used to generate the LFSR-A keystream $(s_{A_0}, .., s_{A_4})$.

Step 2.2: The LFSR-A keystream snippet $(s_{A_0}, s_{A_1}, s_{A_4})$ is subtracted from the generator keystream snippet (s_0, s_1, s_4). This results in two LFSR-B keystream snippets $(s_{B_0}, s_{B_1}, s_{B_{4,0}})$ and $(s_{B_0}, s_{B_1}, s_{B_{4,1}})$. There are two possible values for s_{B_4} because the subtraction's carry-over at index position *3* is unknown. The complexity of the exploit increases by factor 2 from 2^{24} to 2^{24+1}.

Step 2.3: The LFSR-B keystream snippet $(s_{B_0}, s_{B_1}, s_{B_4})$ is mapped to the LFSR-B seed (k_2, k_3, k_4) by a lookup-table: $(k_2, k_3, k_4) := [(s_{B_0}, s_{B_1}, s_{B_4})]_{LFSR\text{-}B}$. This is possible because there is a unique one-to-one mapping from an LFSR-B seed (k_2, k_3, k_4) to the LFSR-B keystream snippet $(s_{B_0}, s_{B_1}, s_{B_4})$.[↳8.7] A lookup-table with the index $(s_{B_0}, s_{B_1}, s_{B_4})$ needs to be calculated beforehand for all possible 2^{24} LFSR-B seeds (k_2, k_3, k_4). This requires additional CPU time and memory. However, the complexity of the exploit does not increase.

Step 3: All the 2^{25} keys k need to be verified. A key k matches if $(k , c) \rightarrow k$.[↳2.4]

Notes:

– The approach presented in this chapter differs slightly from the approach published by Frank A. Stevenson: Instead of guessing (k_0, t_0, t_1), Frank A. Stevenson guesses the values (k_0, k_1, t_0). The latter leads to a slightly more extensive implementation since it requires an additional index table to compute the keystream byte s_1. Anyway, the complexity of both approaches is the same.

– The implementation in libdvdcss[4] is flawed: it considers the first matching disc-key as genuine disc-key. For example, VLC[25] fails to play *"Alien"* (1979, EAN[26]: 4010232109088) since libdvdcss (with `DVD_CSS_METHOD=disc`) uses the first matching disc-key, which is not the genuine disc-key. It requires additional measures to determine the genuine disc-key.[↳6.1] Anyway, the flawed implementation appears to be no problem since libdvdcss uses by default its 30 player-keys to decrypt a feature.

25 http://www.videolan.org/vlc/index.html

26 http://en.wikipedia.org/wiki/International_Article_Number_%28EAN%29

6.5 Player-Key Keystream Exploit

The disc-key decryption is a two-step operation. In the 2nd step, the encrypted disc-key *c* and the generated keystream *s* are fed to the decryptKey-function which returns the decrypted disc-key *p*.↳4.3

The exploit to recover the keystream *s* from an encrypted disc-key *c* and the decrypted disc-key *p* was presented by Frank A. Stevenson[2] and is illustrated in figure 24.

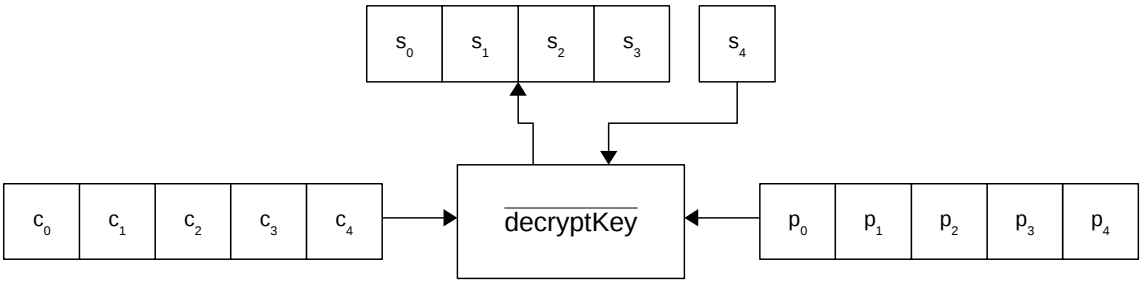


Figure 24: Player-Key Keystream Exploit

The 4-byte keystream snippet (s_0 ,.., s_3) is deduced from *c*, *p* and the keystream byte s_4.

The ten equations of the decryptKey-function are transformed as shown below:

1	$t_4 = s_4 \oplus [\, c_4] \oplus c_3$	(eqn. 1)	6	$t_1 = s_2 \oplus [\, t_2] \oplus p_2$	(eqn. 8)
2	$t_3 = s_4 \oplus [\, t_4] \oplus p_4$	(eqn. 6)	7	$s_1 = t_1 \oplus [\, c_1] \oplus c_0$	(eqn. 4)
3	$s_3 = t_3 \oplus [\, c_3] \oplus c_2$	(eqn. 2)	8	$t_0 = s_1 \oplus [\, t_1] \oplus p_1$	(eqn. 9)
4	$t_2 = s_3 \oplus [\, t_3] \oplus p_3$	(eqn. 7)	9	$s_0 = t_0 \oplus [\, c_0] \oplus t_4$	(eqn. 5)
5	$s_2 = t_2 \oplus [\, c_2] \oplus c_1$	(eqn. 3)	10	$p_0 = s_0 \oplus [\, t_0]$	(eqn. 10)

The equations are applied for all 2^8 possible s_4. Whenever equation number 10 evaluates to true, a matching keystream is found. A survey↳10.15 yielded zero to five matching keystreams for the given set of studied DVD-Videos.

6.6 Keystream Correlation Exploit

The exploit is based on the correlation[27] between the LFSR-A keystream s_A, the LFSR-B keystream s_B and the generator's keystream s.

In normal operation, the keystream bits s_A and s_B are optionally inverted and then fed to the 1-bit full-adder that produces the generator's keystream bit s and the carry-over bit c':

$$(s, c') := (s_A \oplus i_A) + (s_B \oplus i_B) + c$$ ↳4.1

However, if the 1-bit full-adder is replaced by an 1-bit full subtractor[28] (SUB), an LFSR keystream can be recovered from the generator's keystream and the other LFSR keystream. For example, to recover the LFSR-B keystream s_B:

$$(s_B \oplus i_B , c') := s - (s_A \oplus i_A) - c$$

If the LFSR-A register r_A is known, alongside with the (initial) carry-over bit c, and if the 25-bit keystream $(s_0 ,.., s_{24})$ is also known, then it will be possible to recover the LFSR-B register r_B.

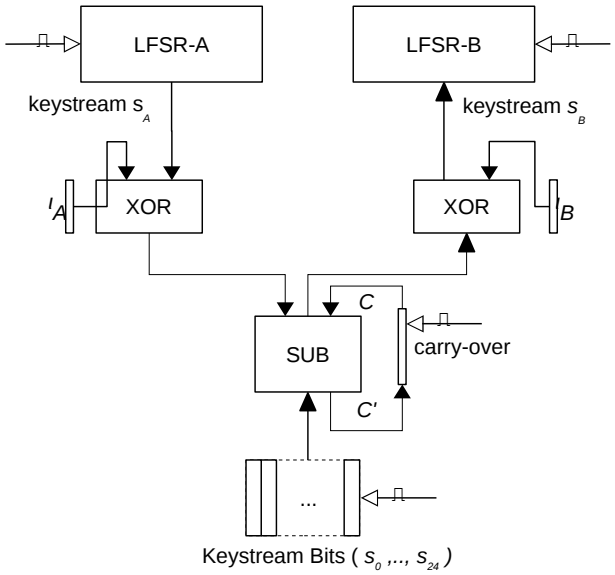


Figure 25: Correlation Attack to Recover LFSR-B

For a given 25-bit keystream-snippet $(s_{n+0} ,.., s_{n+24})$ it would take a brute-force attack to test all $2^{17+25+1}$ generator states. However, with the correlation exploit it is possible to find all matching generators at a complexity of 2^{18} (17-bit LFSR-A register, plus the initial carry-over bit).

For a given 25-bit keystream-head $(s_0 ,.., s_{24})$ it would take a brute-force attack to test all 2^{40} generator start-states. However, with the correlation exploit it is possible to find all matching generators at a complexity of 2^{16}. (Though it appears there are only 2^{15} valid start states since back-shifting the LFSR-B register results half the time in invalid start states.) This exploit was presented by Frank A. Stevenson.[2]

See the next chapter to single out a *genuine* generator. ↳6.7

27 http://en.wikipedia.org/wiki/Correlation_attack

28 http://en.wikipedia.org/wiki/Subtractor#Full_subtractor

6.7 Generator Elimination

A generator state g can be defined as $g = (r_A, r_B, c)$ ↳4.1. With the 17-bit register r_A, the 25-bit register r_B and the 1-bit carry-over bit c, there are 2^{43} different generator states $G = \{ g_i \} \mid i = 0..2^{43}-1$.

However, a keystream generator is set up with a 5-byte seed. The initial value of the carry-over bit is zero. Hence, there are 2^{40} different generator start-states $G_0 \subset G$.

With the keystream correlation exploit ↳6.5 a set of matching generator states can be found for a given 25-bit keystream. Additional keystream bits can be used to single out the genuine generator. That is, all (remaining) generators transition into their next state and the produced keystream bit is compared with the given keystream bit until only one generator remains.

For every generator state g_i there is a unique n-bit keystream head $(s_0 ,.., s_{n-1})$

– where at least one generator state g_j (i ≠ j) exists that produces the same $(n-1)$-bit head $(s_0 ,.., s_{n-2})$
– but where no other generator state g_k (i ≠ k) exists that produces the same n-bit head $(s_0 ,.., s_{n-1})$.

For all generators in *(1,0)-mode* the unique keystream heads were computed as follows:

– For $i \in G_0$ the shortest unique keystream has 35 bits and the longest unique keystream has 78 bits ↳8.9.
– For $i \in G$ the shortest unique keystream has 39 bits and the longest unique keystream has 88 bits ↳8.10.

So,

– to recover a genuine generator start-state in *(1,0)-mode* it requires at least 5 bytes and at most 10 bytes;
– to recover any genuine generator state in *(1,0)-mode* it requires at least 5 bytes and at most 11 bytes.

6.8 LFSR Reversal

In order to recover the seed for a generator, it may be necessary to shift an LFSR[↳4.2] backwards.

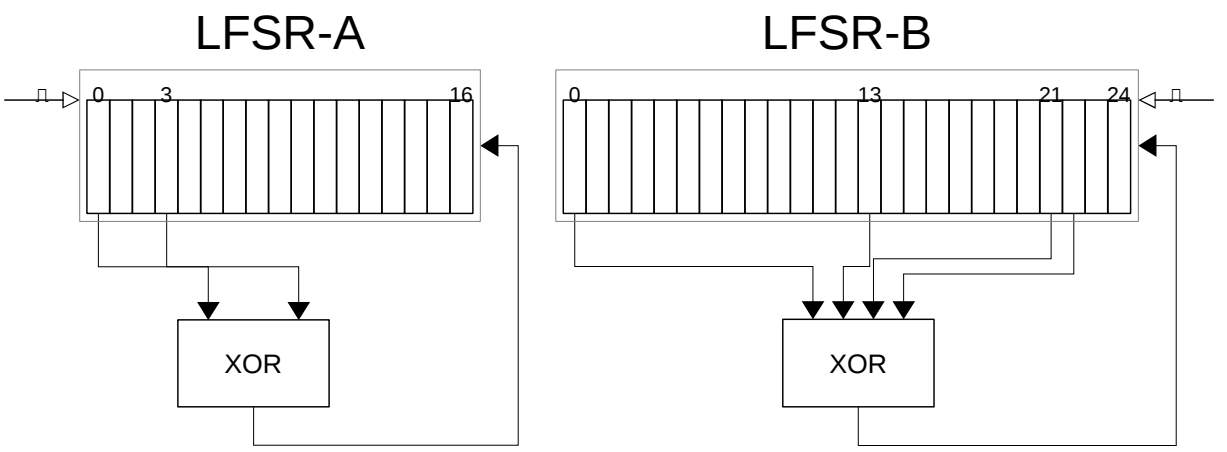


Figure 26: Back-shifting LFSR-A and LFSR-B

Shifting an LFSR backwards is the reversal of the *next*-method. Both LFSRs are driven by XOR of selected register bits. The bit positions that affect the previous state are called the taps. The lowest register bit r_0 is the keystream bit *s*.

LFSR-A taps bits 0 and 3, that is $r.tap() = r_0 \oplus r_3$.

LFSR-B taps bit 0, 13, 21 and 22, that is $r.tap() = r_0 \oplus r_{13} \oplus r_{21} \oplus r_{22}$.

To transition from an LFSR state *r* into the previous LFSR state $r' := r.prev()$:
– the tap is applied,
– the *n*-bit register *r* is left-shifted, that is $r'_i := r_{i+1} \mid i = 0 .. (n-2)$,
– the highest bit of the *n*-bit register is set to the previously tapped value, that is $r'_{n-1} := r.tap()$.
If all bits of register *r* are zero, all of the previous state *r'* bits are also zero. This null-cycle has a period length of 1 state.

For none-zero registers,
– the LFSR-A cycle has a period with 2^{17}-1 different states,[↳8.1]
– the LFSR-B cycle has a period with 2^{25}-1 different states.[↳8.2]

Notes:

– In general, shifting an LFSR can lead to an invalid start-state. That is, a generator is invalid if bit-8 of LFSR-A is not set, if bit-3 of LFSR-B is not set, or, if the carry-over bit of the generator is set.

– If the keystream correlation exploit[↳6.5] for G_0 is used, LFSR-B may end up in an invalid start-state. It appears that only 2^{15} of the produced 2^{16} LFSR-B states are valid.

6.9 Feature Prediction

A challenge for the title-key recovery is the prediction of a plain-text feature snippet. Though, if prediction were easy, we wouldn't need to decrypt the feature in the first place. Luckily, there are a few simple approaches to predict data snippets in a DVD-Pack↳2.9.

[1] The probably most simple and reliable way to predict a snippet is a padding stream (BE_H) which 6-byte header is located in the not-encrypted area before offset 80_H. Then all plain-text bytes from offset 80_H till the end of the DVD-Pack have to be FF_H filler bytes.↳2.10

[2] This is almost the same as the first approach. However, the header of the padding stream can be located in the encrypted area. If there are two elementary streams (PES) inside a DVD-Pack and the 1st PES is either BD_H, $C\#_H$, or $E0_H$ then the 2nd PES appears to be a padding stream (BE_H).[10.8] So the plain-text consists of the four bytes with the stream-id (00_H,00_H,01_H,BE_H), two bytes with the PES length (the length results from the size of the 1st PES), and finally the FF_H filler bytes till the end of the pack. – This kind of prediction is mentioned in libdvdcss[4], but was never fully implemented.

[3] The video-data of the feature is stored as compressed pixel-blocks. There is a certain likelihood that a pixel-block *w* (word) repeats. Even several times forming a cycle *ww*+[29]. If such a cycle crosses over offset 80_H, it can be used for prediction: The part of the word *w* before the 80_H-border is the head *h*, the part at the 80_H-border is the tail *t*, so the cycle *ww*+ becomes *(ht)(ht)*+. If the not-encrypted part, before the 80_H-border, contains any pattern *h(tw*∗*)h*, then the prediction after the 80_H-border is *tw*+. – This kind of prediction, in a more conservative variant, was first presented by Ethan Hawke[3]. Libdvdcss[4] uses another variant of this kind of prediction.

In order to identify uniquely a generator start-state, a 35- to 78-bit keystream head is required. A 39- to 88-bit keystream snippet is required if its not a head keystream. So, five bytes may be sufficient, but ten bytes will always do for the approaches [1] and [3]. Eleven bytes will always do for approach [2].

Unfortunately it appears↳10.20 that not all VMG/VTS contain at least one encrypted DVD-Pack with a padding stream. Searching for a cycle is more promising. Almost all VMG/VTS appear↳10.21 to contain at least one DVD-Pack with an easy to predict cycle.

Notes:

– The encryption of a DVD-Pack is optional. So, if the manufacturer decides not to encrypt vulnerable packs which contain predictable data then it won't be possible to recover title-keys. Though, there are probably many methods of prediction besides those presented here.

– The libdvdcss prediction is kind of strange. It expects the repetition of a word *w*, however the last character of the second word is never checked. Also, the supposed plain-text might contain cipher-text. Anyway, whether this is intended or a bug, it is certainly still effective.

29 http://en.wikipedia.org/wiki/Regular_expression#POSIX_extended

7 Test Program

The accompanied software is available as source code and hosted on GitHub.

The source code is C++ and written for GNU/Linux. The implementation makes use of the Linux SCSI Generic (SG) driver to access a DVD drive.

The program implements two keystream generators. There is a bit-stream generator as described in this documents. And there is a faster byte-stream generator that is similar to the one used by libdvdcss[4].

Compiled with GCC 4.7.2 (Debian 4.7.2-5) target: x86_64-linux-gnu, GNU Make 3.81, libboost 1.49.0 on Intel Celeron CPU G530.

Compiled with GCC 4.6.3 (Debian 4.6.3-14+rpi1) target: arm-linux-gnueabihf, GNU Make 4.0, libboost 1.49 on ARMv7 Processor rev 5 (v7l).

For further analysis awk and perl scripts have also been used.

The user interface is command line based.

Synopsis: css_study command option arguments

The list of available commands is provided by: css_study help

The list of available options is provided by: css_study command help

The list of available arguments is provided by: css_study command option help

The purpose of the program is pure academically.

8 Test Suit (Cryptanalysis)

8.1 LFSR-A Period

If the start value is 0, the size of the period is 1. If the start value is not 0, the size of the period is 131,071, which is 2^{17}-1.

Test-Case: lfsr-a shift value [--verbose 0|1]

Print the register value and transition into the next LFSR-A state[↳4.2]. Repeat until the cycle is completed.

Test-Case: lfsr-a unshift value [--verbose 0|1]

Print the register value and transition into the previous LFSR-A state[↳6.8]. Repeat until the cycle is completed.

8.2 LFSR-B Period

If the start value is 0, the size of the period is 1. If the start value is not 0, the size of the period is 33,554,431, which is 2^{25}-1.

Test-Case: lfsr-b shift value [--verbose 0|1]

Print the register value and transition into the next LFSR-B state[↳4.2]. Repeat until the cycle is completed.

Test-Case: lfsr-b unshift value [--verbose 0|1]

Print the register value and transition into the previous LFSR-B state[↳6.8]. Repeat until the cycle is completed.

8.3 Disc-Key Study

On an Intel Celeron CPU G530 @ 2.40 GHz it would take a single thread approximately 24 hours to test all keys. A single thread on an ARMv7[30] CPU @ 0.9 GHz would need approximately 145 hours. An optimized implementation might be considerably faster though.

Test Case: study disc-key hash

Decrypt all possible 2^{40} disc-keys with the given hash-value and report if a result matches.[↳6.1]

8.4 Disc-Key Exploit

On an Intel Celeron CPU G530 @ 2.40 GHz it takes a single thread approximately 4 seconds to recover all matching keys. A single thread on an ARMv7 CPU @ 0.9 GHz needs approximately 15 seconds.

Test Case: recover disc-key hash

Recover all matching disc-keys for the given hash-value.[↳6.1]

30 Raspberry Pi 2 Model B

8.5 Player-Key Study

On an Intel Celeron CPU G530 @ 2.40GHz it would take a single thread approximately 24 hours to test all keys. A single thread on an ARMv7 CPU @ 0.9 GHz would need approximately 145 hours. An optimized implementation might be considerably faster though.

Test Case: study player-key disc-key cipher

Decrypt the given cipher with all possible 2^{40} player-keys and report if the decrypted result is the given disc-key.[↳6.2]

8.6 Player-Key Exploit

An Intel Celeron CPU G530 @ 2.40GHz as an ARMv7 CPU @ 0.9 GHz need only a fraction of a second to recover all matching keys.

Test Case: recover player-key disc-key cipher

Recover all matching player-keys for the given cipher and disc-key.[↳6.2]

8.7 LFSR-B Seed / Keystream Correlation

Test Case: lfsr-b stream

The test case produces for every 3-byte seed (k_2, k_3, k_4) a 5-byte keystream $(s_0 ,.., s_4)$ for LFSR-B.[↳4.2]

The table lists for each 3-byte keystream-snippet-combination the number of unique values. A combination is considered unique (marked as ✓) if it covers all possible 2^{24} 3-byte values. In this case there is a one-to-one mapping for each 3-byte keystream-snippet to the 3-byte seed (k_2, k_3, k_4). Such one-to-one mapping is required to perform the disc-key exploit for the keystream-snippet (s_0, s_1, s_4).[↳6.4]

Combination			Unique Values	
s_0	s_1	s_2	16,777,216	✓
s_0	s_1	s_3	1,048,576	-
s_0	s_1	s_4	16,777,216	✓
s_1	s_2	s_3	16,777,216	✓
s_1	s_2	s_4	1,048,576	-
s_2	s_3	s_4	16,777,216	✓

8.8 Unique (0,0)-Mode Keystream Heads (with a valid start state)

Test Case: spread sum 00:00:00 ff:ff:ff --invertA 0 --invertB 0 --valid 1

The distribution of the occurrence of all 2^{40} unique *n*-bit keystreams[↳6.7] for a generator g_i in *(0,0)*-mode with $i \in G_0$ was computed as follows. Note that a single thread on an Intel Celeron CPU G530 @ 2.40 GHz would need approximately 140 hours for the computation.

n	Occurrence	*n*	Occurrence	*n*	Occurrence	*n*	Occurrence
35	1	46	16,926,497,632	57	8,286,261	68	4,178
36	117,936	47	8,582,857,523	58	4,137,797	69	2,226
37	355,436,487	48	4,247,999,886	59	2,048,012	70	1,024
38	19,690,625,748	49	2,139,403,852	60	1,029,062	71	586
39	128,738,762,540	50	1,061,218,157	61	546,934	72	302
40	256,055,836,512	51	529,141,036	62	263,560	73	120
41	262,899,628,144	52	265,492,401	63	125,504	74	74
42	188,968,309,105	53	132,486,478	64	64,634	75	42
43	113,687,079,204	54	65,886,365	65	32,058	76	28
44	62,497,288,445	55	33,068,320	66	16,166	77	6
45	32,601,371,315	56	16,553,765	67	8,346	78	4

The shortest unique keystream consists of 35 bits:

– The seed (10_H,$B8_H$,$E1_H$,49_H,BA_H) results in the keystream (FF_H,FF_H,$8F_H$,FF_H,$1B_H$).

The longest non-unique keystream consists of 77 bits. There are two such keystreams:

– The seed (51_H,$C5_H$,17_H,CB_H,37_H) results in the keystream (81_H,89_H,AB_H,$A6_H$,89_H,$3A_H$,$F6_H$,36_H,ED_H,61_H).

– The seed ($B9_H$,$B7_H$,$C4_H$,97_H,94_H) results in the keystream (81_H,89_H,AB_H,$A6_H$,89_H,$3A_H$,$F6_H$,36_H,ED_H,81_H).

– The seed (32_H,$D7_H$,$E5_H$,10_H,32_H) results in the keystream (92_H,$9E_H$,79_H,25_H,$9B_H$,$A8_H$,$9D_H$,64_H,AE_H,62_H).

– The seed ($F6_H$,$D5_H$,65_H,DE_H,$F8_H$) results in the keystream (92_H,$9E_H$,79_H,25_H,$9B_H$,$A8_H$,$9D_H$,64_H,AE_H,$c2_H$).

That means, a 35-bit keystream may be sufficient to uniquely identify a generator. However, a 78-bit keystream is required in the worst case.

8.9 Unique (1,0)-Mode Keystream Heads (with a valid start state)

Test Case: spread sum 00:00:00 ff:ff:ff --invertA 1 --invertB 0 --valid 1

The distribution of the occurrence of all 2^{40} unique *n*-bit keystreams[↳6.7] for a generator g_i in *(1,0)*-mode with $i \in G_0$ was computed as follows. Note that a single thread on an Intel Celeron CPU G530 @ 2.40 GHz would need approximately 140 hours for the computation.

n	Occurrence	*n*	Occurrence	*n*	Occurrence	*n*	Occurrence
35	1	46	16,926,467,275	57	8,286,258	68	4,178
36	117,739	47	8,582,857,606	58	4,137,779	69	2,226
37	355,451,181	48	4,247,997,201	59	2,048,006	70	1,024
38	19,690,475,588	49	2,139,404,180	60	1,029,062	71	586
39	128,738,755,565	50	1,061,216,423	61	546,934	72	302
40	256,056,799,238	51	529,140,229	62	263,560	73	120
41	262,899,159,006	52	265,492,032	63	125,504	74	74
42	188,968,484,329	53	132,486,520	64	64,634	75	42
43	113,686,736,885	54	65,886,656	65	32,058	76	28
44	62,497,189,144	55	33,068,428	66	16,166	77	6
45	32,601,321,930	56	16,553,723	67	8,346	78	4

The shortest unique keystream consists of 35 bits:

– The seed (13_H,$8B_H$,$F9_H$,$C3_H$,$C0_H$) results in the keystream (FF_H,FF_H,$8F_H$,FF_H,DF_H).

The longest non-unique keystream consists of 77 bits. There are two such keystreams:

– The seed (51_H,$C5_H$,$C4_H$,97_H,94_H) results in the keystream (40_H,$3D_H$,$A1_H$,$0D_H$,87_H,$C1_H$,$D1_H$,$0D_H$,AC_H,$C9_H$).

– The seed ($B9_H$,$B7_H$,17_H,CB_H,37_H) results in the keystream (40_H,$3D_H$,$A1_H$,$0D_H$,87_H,$C1_H$,$D1_H$,$0D_H$,AC_H,$A9_H$).

– The seed (32_H,$D7_H$,65_H,DE_H,$F8_H$) results in the keystream ($C8_H$,96_H,$7B_H$,$A1_H$,DC_H,FF_H,20_H,$3C_H$,$B6_H$,04_H).

– The seed ($F6_H$,$D5_H$,$E5_H$,10_H,32_H) results in the keystream ($C8_H$,96_H,$7B_H$,$A1_H$,DC_H,FF_H,20_H,$3C_H$,$B6_H$,$A4_H$).

That means, a 35-bit keystream may be sufficient to uniquely identify a generator. However, a 78-bit keystream is required in the worst case.

8.10 Unique (1,0)-Mode Keystream Snippets (with any start state)

Test Case: spread sum 00:00:00 ff:ff:ff --invertA 1 --invertB 0 --valid 0

The distribution of the occurrence of all 2^{43} unique *n*-bit keystreams[↳6.7] for a generator g_i in *(1,0)*-mode with $i \in G$ was computed as follows. Note that a single thread on an Intel Celeron CPU G530 @ 2.40 GHz would need approximately 89 days for the computation.

n	Occurrence	*n*	Occurrence	*n*	Occurrence	*n*	Occurrence
39	897,381	52	17,020,924,412	65	2,098,400	78	258
40	2,884,305,481	53	8,521,703,685	66	1,039,510	79	112
41	157,614,056,282	54	4,258,327,404	67	518,392	80	38
42	1,029,898,440,795	55	2,132,367,614	68	259,300	81	24
43	2,046,259,159,836	56	1,069,926,770	69	130,296	82	16
44	2,098,034,657,652	57	534,965,697	70	65,424	83	12
45	1,514,670,922,228	58	267,115,519	71	32,872	84	6
46	911,742,449,860	59	133,448,661	72	16,470	85	6
47	500,822,211,152	60	66,626,314	73	8,480	86	4
48	263,205,033,162	61	33,372,808	74	4,242	87	4
49	134,944,113,329	62	16,721,314	75	2,204	88	2
50	67,978,311,025	63	8,384,933	76	1,168		
51	33,966,201,037	64	4,200,063	77	554		

The shortest unique keystream consists of 39 bits. There are 897,381 different generators. For example the seed ($1BCC1_H$,$41E74E_H$,0) results in the unique keystream (00_H,00_H,00_H,FC_H,$3F_H$).

The longest non-unique keystream consists of 87 bits.

– The seed (1050_H,$1C3DE11_H$,0) results in the keystream ($6F_H$,66_H,BF_H,$F6_H$,BA_H,AF_H,01_H,62_H,$A2_H$,$6E_H$,DA_H).

– The seed ($22CE_H$,$1D34FFB_H$,1) results in the keystream ($6F_H$,66_H,BF_H,$F6_H$,BA_H,AF_H,01_H,62_H,$A2_H$,$6E_H$,$5A_H$).

That means, a 39-bit keystream may be sufficient to uniquely identify a generator. However, an 88-bit keystream is required in the worst case.

9 Drive Survey

This survey regards ten not representatively selected consumer drives.

9.1 Drives

Test Case: mmc inquiry device

Test Case: mmc report-rpc device

#	Vendor-ID	Drive-ID	Revision	Vendor	Interface	Region
1	TSSTcorp	DVD-ROM TS-H352A	TS03	Samsung	PATA	2
2	HL-DT-ST	DVD-ROM GDR8164B	0L06	LG	PATA	2
3	HL-DT-ST	DVDRAM GSA-4163B	A103	LG	PATA	2
4	TOSHIBA	DVD-ROM SDM2012C	TU01	Toshiba	PATA	2
5	ATAPI	iHOS104	WL0F	Lite-On	SATA	2
6	Slimtype	eTAU108 1	EL46	Lite-On	USB 2.0	2
7	PLDS	DVD-RW DS8ABSH	AL61	Lite-On	SATA (Slimline)	2
8	Optiarc	DVD+-RW AD-7700H	100A	Sony	SATA (Slimline)	2
9	MATSHITA	DVD-RAM UJ-841S	1.00	Panasonic	PATA (Slimline)	2
10	HL-DT-ST	DVD-RW GWA-4080N	0C09	LG	PATA (Slimline)	2

All drives come with RPC-2.[↳3.4]

Note: Linux (kernel 3.2.0) appears to use always the size of the first inserted DVD on a USB drive (eTAU108). That is, if another DVD is (manually) inserted, the read[31] system call will still be bound to the size of the first DVD, unless the device is manually re-mounted. That may result in odd kinds of errors.

31 http://man7.org/linux/man-pages/man2/read.2.html

9.2 AGID Management

Test Case: mmc report-agid device

An AGID is used as a unique session identifier for handshakes.[↳3.1,3.2] Four AGID numbers are defined: 0..3.[8] The host requests an AGID from the drive, and the AGID remains in use until the session completes, or until the AGID is invalidated.

An AGID is abandoned if a host doesn't invalidate an AGID for an uncompleted-but-no-longer-used session (by intention or fault). The drive doesn't reclaim abandoned AGIDs by timeout. At a certain point there is no AGID available any more if all of them were abandoned.

Another problem is the non-existent ownership of an AGID. It is not possible to identify the process that allocated an AGID. As a consequence it's not possible to detect an abandoned AGID that has been left behind after a process terminated. Note that any process can invalidate any (still-in-use or abandoned) AGID.

This makes it virtually impossible to take the AGID management seriously.

Moreover, many drives manage AGIDs incompletely. The worst drive did always return AGID number 0 for each request, regardless whether the AGID was in use or not.

There were only a few drives in the survey that operated sensible. That is: they provided all four AGID numbers, they returned a reasonable error code when no more AGIDs were available, and they didn't reclaim any AGID that might still be in use.

Actually most drives support only one AGID number (which is actually no problem). On a second allocation request they respond with a *"command sequence error"* (sense code $2C00_H$) which is still fine (although reporting a resource problem might be better). At the same moment however, these drives invalidate the AGID on their own, even if the AGID is still in-use.

Furthermore it appears that all drives (which support only one AGID number) ignore the AGID number that is actually used in a request.

#	Range	If no more AGIDs are available, the drive returns...	
1	3	...*"command sequence error"* (sense code $2C00_H$) and invalidates the AGID	✗
2	0	...always the same AGID (reports never an error)	✗
3	0..3	...*"system resource failure"* (sense code 5500_H)	✓
4	3	...*"command sequence error"* (sense code $2C00_H$) and invalidates the AGID	✗
5	0	...the same AGID. (reports no error on 1st time)	✗
		...*"key not present"* (sense code $6F01_H$) and invalidates the AGID (after 1st time)	✗
6	3	...*"command sequence error"* (sense code $2C00_H$) and invalidates the AGID	✗
7	3	...*"command sequence error"* (sense code $2C00_H$) and invalidates the AGID	✗
8	3	... no description (sense code 0000_H)	✓

#	Range	If no more AGIDs are available, the drive returns...	
9	3	...*"system resource failure"* (sense code 5500_H)	✓
10	0..3	...*"system resource failure"* (sense code 5500_H)	✓

Note: libdvdcss[4] sends MMC commands to invalidate all four AGID numbers if an AGID allocation fails and tries then again. This appears to be reasonable, considering the circumstances.

9.3 Copyright Management Information and Title-Key

Test Case: handshake title device lba

The table below lists the result of a title-key retrieval↳3.2 depending on the value of the CPR_MAI field.

#	CPR_MAI = 00_H	CPR_MAI = $B0_H$	CPR_MAI = $F0_H$
1	title-key	title-key	title-key
2	sense code $6F01_H$	title-key	title-key
3	sense code $6F01_H$	title-key	title-key
4	title-key	title-key	title-key
5	title-key	title-key	title-key
6	title-key	title-key	title-key
7	title-key	title-key	title-key
8	title-key	title-key	title-key
9	title-key	title-key	title-key
10	sense code $6F01_H$	title-key	title-key

Sense code $6F01_H$: *"copy protection key exchange failure - key not present"*

9.4 Title-Key without Authentication

Test Case: handshake title device lba

A title-key handshake↳3.2 could be executed without an authentication handshake before. Almost all drives support this. However, interestingly, some drives deliver the title-key without the buskey obfuscation for the first handshake.

#	1st time	2nd time
1	not obfuscated	obfuscated
2	obfuscated	obfuscated
3	obfuscated	obfuscated
4	not obfuscated	obfuscated
5	obfuscated	obfuscated
6	not obfuscated	obfuscated
7	not obfuscated	obfuscated
8	obfuscated	obfuscated
9	Sense code $6F00_H$	Sense code $6F00_H$
10	obfuscated	obfuscated

* Sense code $6F00_H$: *"copy protection key exchange failure - authentication failure"*

9.5 Handshakes for Non-DVD-Video

Test Case: --log 3 handshake unlock device

Test Case: --log 3 handshake title device lba

The table below lists the handshake[↳3.1,3.2] results for a DVD which is no DVD-Video.

#	Authentication-Handshake	Title-Key-Handshake
1	command AD:02 at step (6) key not present ($6F01_H$)	(00,00,00,00,00)
2	command A4:00 at step (1) incompatible format (3002_H)	same as for authentication
3	command A4:02 at step (3) key not present ($6F01_H$)	same as for authentication
4	command A3:01 at step (2) timeout	same as for authentication
5	command AD:02 at step (6) key not present ($6F01_H$)	(00,00,00,00,00)
6	command AD:02 at step (6) key not present ($6F01_H$)	(00,00,00,00,00)
7	command AD:02 at step (6) key not present ($6F01_H$)	(00,00,00,00,00)
8	command A4:00 at step (1) no description (0000_H)	same as for authentication
9	command A4:00 at step (1) incompatible format (3002_H)	same as for authentication
10	command A4:00 at step (1) incompatible format (3002_H)	same as for authentication

9.6 Region-Code Mismatch

Test Case: handshake unlock device

Test Case: mmc read device --cluster 32

All drives of the survey (with permitted region 2) accept the authentication-handshake[↳3.1] even if the region-code of the DVD-Video (permitted region 1) does not match. All drives but one unlock locked feature regions, so the complete (encrypted) feature can be read on those drives.

	Time to read the complete feature
1	15:33 minutes w/o errors
2	26:44 minutes w/o errors
3	17:23 minutes w/o errors
4	16:20 minutes w/o errors
5	17:37 minutes w/o errors
6	2:25:06 hours with 589 errors (sense code 0200_H and 1105_H)
7	17:11 minutes w/o errors
8	35:49 minutes w/o errors
9	Does not read locked regions: *"media region code is mismatched to logical unit region"* ($6F04_H$)
10	34:49 minutes w/o errors

Test Case: handshake title-key device lba count

However, none of the drives accepts the title-key-handshake[↳3.2]. All drives return an error message in the handshake at step (6).

For drive #8 it is: *"copy protection key exchange failure - key not present"* (sense code: $6F01_H$).

For all other drives it is: *"media region code is mis-matched to logical unit region"* (sense code: $6F04_H$).

9.7 Title-Key and CPR_MAI Throughput

Test Case: handshake title device lba count

Test Case: mmc cmi device lba count

Reading the title-keys[↳2.6] from all frames takes a long time. The table below shows the time in seconds to retrieve 1,000 title-keys from the begin, middle and end of a randomly chosen DVD-Video. The right side of the table shows the time to retrieve just the copyright management information[↳2.2] (CPR_MAI w/o title-key).

	Title-Key				CPR_MAI		
	Begin	Middle	End		Begin	Middle	End
1	36.469	32.380	31.326		-*	-*	-*
2	86.481	58.908	58.741		57.559	29.904	29.672
3	50.561	38.450	38.373		38.728	37.572	25.716
4	44.341	39.133	37.981		44.206	38.225	38.118
5	74.388	72.560	72.445		51.814	45.598	45.522
6	59.829	57.037	58.506		57.128	50.920	52.574
8	108.043	56.238	52.824		105.457	52.956	52.802
9	10.560	10.026	9.832		9.947	7.023	6.750
10	77.888	51.516	51.352		78.282	51.509	49.790

* Sense code 2400_H: *"invalid field in cdb"*

10 DVD Survey

This survey regards 244 not representatively selected consumer DVD-Videos.

The following two (not CSS-encrypted) DVD-Videos are explicitly listed (and referred to below) since they show some kind of odd behavior:

– A Chinese Ghost Story (1987), EAN 4026621350771

– A Chinese Ghost Story 2 (1990), EAN 4026621350788

10.1 Copyright Information

Test Case: mmc copyright device

145 DVD-Videos are marked as CSS encrypted. 99 DVD-Videos are not marked as CSS-encrypted. ↳2.2

10.2 Region-Code

Test Case: mmc copyright device

The table below lists the occurrence of region-codes: ↳2.3

Mask	Permitted Regions								Occurrence[*1]	Occurrence[*2]
	1	2	3	4	5	6	7	8		
00_H	✓	✓	✓	✓	✓	✓	✓	✓	4	33
40_H	✓	✓	✓	✓	✓	✓		✓	5	13
$7D_H$		✓						✓	8	8
$C0_H$	✓	✓	✓	✓	✓	✓			1	13
$C1_H$		✓	✓	✓	✓	✓			0	1
$E5_H$		✓		✓	✓				19	19
ED_H		✓			✓				14	14
$F5_H$		✓		✓					16	17
FD_H		✓							77	98
FE_H	✓								1	1

[*1] The 145 CSS-encrypted DVD-Videos, [*2] All 244 DVD-Videos

10.3 Copyright Management Information

Test Case: mmc cmi device lba [--count lba]

The Copyright Management Information (CPR_MAI)↳2.5 field on a DVD-Video appears to hold always one of the following three values:

– 00_H: Not copyrighted.

– $B0_H$: No copying is allowed.

– $F0_H$: No copying is allowed and the DVD-Pack is scrambled by CSS with a title-key.

Note: Only about 20 DVD-Videos have been checked since reading all CPR_MAI fields takes a long time.

10.4 ISO Integrity

Test Case: iso list file

All DVD-Videos support the DVD bridge disc format.↳2.7

The two Chinese Ghost Story DVDs use the invalid ISO file id 0 (zero). Permitted values are only $1..2^{31}-1$.[6]

10.5 Maximum File Size

Test Case: feature list file

On one DVD-Video there is one VOB file with a size of 2^{30} bytes which is above the permitted size of $2^{30}-800_H$.↳2.8

10.6 Feature Contiguity and Integrity

Test Case: feature list file

Test Case: ifo list file

Test Case: ifo verify file

The feature's integrity is supposed to be met if the VIDEO_TS.IFO's VMG and VTS information are consistent and align with the information of the file records.↳2.8

The two Chinese Ghost Story DVDs set the VMG IFO *bupL* value to the last VOB LBA and not to the last BUP LBA. Even so, the file records (VIDEO_TS.IFO, VIDEO_TS.BUP) hold the correct LBA- and size values.

Five DVD-Videos (all of them extras, issued by the same magazine) set the VTS *bupL* value to the last VOB LBA plus the number of IFO blocks. That would be correct if there wasn't a 16-block gap between VOB-end and BUP-start. Interestingly, all the VMG *bupL* values are correct, even though there is the same 16-block gap. Anyway, all file records hold the correct LBA- and size values.

Technically, it appears to be of no problem to glue VMG and VTS areas together without any gap. Also, there need to be no gap between the IFO, VOB and BUP areas. 148 DVD-Videos do so and contain no gap at all. The remaining 96 DVD-Videos contain at least one gap. Various of them align a file's first block to a by-32-

divisible LBA (but not the VOB VTS file extensions). Others leave a 16-block gap between VOB-end and BUP-start.

10.7 Optional VOB Menus

Test Case: feature list file

Test Case: ifo list file

The VMG and VTS VOB menus are optional.[↳2.8]

Only three DVD-Videos hold no VMG VOB menu. That is, the VMG IFO *vobF* value is set to zero and the VIDEO_TS.VOB file-record is omitted.

There are 1,293 VTS on the 244 DVD-Videos. 150 VTS contain no VOB menu. For 108 of them the file-record is omitted. For the remaining 42 the file-record shows a zero file-size.

10.8 PES ID

Test Case: pack id decrypted-file

A DVD-Pack is supposed to contain one or two PES.[↳2.9]

The table below lists the occurrence of PES identifiers for all DVD-Packs.

1st PES	2nd PES	Occurrence
BD_H	-	89,594,084
BD_H	BE_H	2,548,043
BE_H	-	84*
BF_H	BF_H	4,224,023
$C0_H$	-	240,529
$C0_H$	BE_H	47,243
$C1_H$	-	182,438
$C1_H$	BE_H	37,740
$C2_H$	-	182,430
$C2_H$	BE_H	37,738
$C3_H$	-	95,364
$C3_H$	BE_H	23,745
$C4_H$	-	95,364
$C4_H$	BE_H	23,745
$E0_H$	-	657,069,787
$E0_H$	BE_H	4,368,579

* Note, the two "Chinese Ghost Story" DVD-Videos contain various DVD-Packs which consist of single padding PES (BE_H) only.

10.9 Number of Encrypted DVD-Packs

Test-Case: pack scrambling file

For the 99 not-encrypted DVD-Videos, the (MPEG-2-PS) scrambling control field[↳2.10] is never set.

For the 145 encrypted DVD-Videos 142,073,477 out of 471,461,848 DVD-Packs are scrambled (30%); ranging for individual DVD-Videos between 11% and 47%.

The encrypted DVD-Videos hold altogether 867 VMG and VTS. 770 VMG+VTS have at least one scrambled DVD-Pack. The remaining 97 VMG+VTS are not scrambled (87 VMG and 10 VTS).

10.10 Scrambling Control Field on Offset 14_H

Test Case: pack scrambling file

The DVD-Video's scrambling control field[↳2.10] is supposed to be located on offset 14_H in a DVD-Pack. However, for various DVD-Packs on the two "A Chinese Ghost Story" DVD-Videos, the payload starts already at offset 14_H. So they indicate sometimes a CSS encryption that is actually not present.

For all other 243 DVD-Videos, the payload does not start before offset 17_H. If the (MPEG-2-PS) scrambling control field is present, it is always located at offset 14_H. If no (MPEG-2-PS) scrambling control field is present, the masked value (30_H) on offset 14_H is always zero.

10.11 Scrambling Control Field and Copyright Management Information

Test Case: mmc cmi device
Test Case: pack scrambling file

The scrambling control field[↳2.10] corresponds to the copyright management information (CPR_MAI)[↳2.5]. That is, the DVD-Pack is scrambled if: CPR_MAI & CF_H = $F0_H$. This also means there is a title-key present.

Note: Only about 20 DVD-Videos have been checked since reading all CPR_MAI fields takes a long time.

10.12 Locked Blocks

Test Case: mmc read device

Unless authenticated the drive denies access to read-protected blocks of the feature.[↳3.1] That is, access is denied if the copyright management information (CPR_MAI)[↳2.5] is: CPR_MAI & CF_H = $F0_H$. Read access fails with: *"read of scrambled sector without authentication"* (sense code $6F03_H$).

Note: Only about 20 DVD-Videos have been checked since reading all CPR_MAI fields takes a long time.

10.13 Disc-Key Hash-Value

Test Case: handshake key device

Test Case: recover disc-key hash-value

The first entry of a disc-key-block ↳2.4 holds the hash-value. There may exist several matching disc-keys for a single hash-value. ↳6.1 For the examined DVD-Videos the number of matching disc-keys ranges between 1 and 5 as listed below.

Matching Keys	Occurrence
1	50
2	53
3	26
4	14
5	2

So about 34% of the DVD-Videos have one matching disc-key, the rest has more than one matching key. This is similar to the result of the keystream analysis ↳8.8 where 37% of the unique keystreams are 40 bits and less, the rest has more than 40 bits.

Note: If there is more than one matching disc-key, the genuine disc-key needs to be singled out. ↳10.16

10.14 Number of Player Keys

Test Case: handshake key device

A disc-key-block ↳2.4 holds 408 slots for player-key-encrypted disc-key-variants. For the examined DVD-Videos there are many identical values per disc-key-block. Actually there are three types of disc-key-blocks as listed below.

Number of Different Values	Number of DVDs
32	10
33	103
64	32

Only 32 different values are used on ten DVD-Videos. Thus, there can't be more than 32 different player-keys for these DVD-Videos. (Actually it means there can't be more than 32 different 5-byte keystreams. ↳6.5) That reduces significantly the number of 408 possible player-keys (a key for each slot).

Note: it would be possible to forge several sets of player-keys, each set for a range of DVD-Videos. However it appears, this did never happen. There are several player-keys which can be applied to every DVD-Video. ↳10.17

10.15 Player-Key Keystream

Test Case: recover title-key disc-key cipher

We use the 1-to-5 matching disc-keys[↳10.13] *p* of the 145 encrypted DVD-Videos and use them to decrypt the 32/33/64 disc-key-variants[↳10.14] *c* of the corresponding disc-key-blocks. This results in 12,240 combinations:

Number of disc-key-variants[↳10.14]	Number of matching disc-keys[↳10.13]						Combinations
	1	2	3	4	5	∑	
32	4	6				10	512
33	39	36	17	9	2	103	6,864
64	7	11	9	5		32	4,864
							12,240

The exploit to recover the keystream is applied for each of the 12,240 *(p,c)* combinations. This results in zero to five keystreams per *(p,c)* combination:

Number of Keystreams (with any start-state)	Occurrence
0	2,052
1	5,327
2	3,674
3	1,008
4	165
5	14
	12,240

Number of Keystreams (with a valid start-state)	Occurrence
0	3,094
1	5,391
2	3,281
3	424
4	48
5	2
	12,240

All the matching disc-keys were used to run the exploit, not only genuine disc-keys. This causes a high number of 2,052 (3,094) cases without any keystream. At this point, the genuine disc-keys are not singled out yet.

10.16 Genuine Disc-Keys

Test Case: recover player-key disc-key cipher

If there is more than one matching disc-key for a hash-value[↳10.13] then the genuine disc-key needs to be singled out. This can be done through player-key recovery. That is, each matching disc-key is used to recover all player-key-encrypted disc-key-variants of the disc-key-block.[↳6.2] Not-genuine disc-keys might result in a high number of not-matching keystreams.[↳10.14]

There are 1-to-5 matching disc-keys for a DVD-Video's hash-value[↳10.13] and there are 32-to-64 different disc-key-variants per block.[↳10.14]

For each disc-key the number of successfully recovered keystreams[↳10.15] is saved (only those keystreams are counted that have a valid generator start-state[↳6.7]). The highest number is listed (best match) in the table below. If there is more than one matching disc-key, the 2nd highest number is also listed (2nd best).

Variants	Best Match	2nd best	Occurrence
32	32	-	4
32	32	≤ 16	6
33	32	-	19
33	32	≤ 24	27
33	33	-	20
33	33	≤ 25	37
64	64	-	7
64	64	≤ 39	25

There is a significant difference between the best match and the 2nd best match. Hence, it appears that the keystream recovery[↳6.5] is suitable to single out the genuine disc-key if several disc-keys match a hash-value[↳6.1].

The table lists also an interesting detail: there are 46 disc-key-blocks with 33 different variants but only 32 matching keystream. One of the variants is a fake.

10.17 Player-Keys

Test Case: recover title-key disc-key cipher

The 145 genuine disc-keys[↳10.17] are used to recover all keystreams with a valid generator's start-state. There are 5.767 player-key-encrypted disc-key-variants on the 145 DVD-Videos.[↳10.14] They result in 1.810 different keystreams as listed below.

Number of DVDs	Common Keystreams	Number of DVDs	Common Keystreams	Number of DVDs	Common Keystreams	Number of DVDs	Common Keystreams
145	31	18	1	10	3	4	23
144	1	15	1	9	1	3	106
33	1	14	3	8	3	2	133
32	31	13	4	7	5	1	1,384
23	1	12	3	6	29		
20	1	11	2	5	43		

There are 31 keystreams which can be used to decrypt the disc-key of all 145 DVD-Videos. Another keystream (actually the one that was used for DeCSS[32]) can be used to decrypt 144 DVD-Videos.

32 http://en.wikipedia.org/wiki/DeCSS

Notes:

– Keystreams are listed instead of player-keys since a keystream with a valid generator's start- state can be produced by several (equivalent) player-keys.

– libdvdcss[4] uses only 30 player-keys (which produce different keystreams) including the DeCSS player-key.

10.18 Disc-Key Distribution

With the high number of possible 5-byte disc-keys, one could expect a uniform disc-key distribution over all DVD-Video releases. However, interestingly, the genuine ↳10.16 disc-keys are not evenly distributed. There are 45 disc-keys starting with the 4-byte sequence (00_H,00_H,00_H,00_H). Though 32 of them belong to the same TV show. However, also various of the remaining disc-keys are used more than once: one disc-key even six times on unrelated DVD-Videos.

Disc-Key (00_H,00_H,00_H,00_H,$\#\#_H$)

Disc-Keys	Occurrence
35	1
2	2
2	3

Remaining Disc-Keys

Disc-Keys	Occurrence
63	1
6	2
3	3
2	5
1	6

10.19 Title-Keys

Test Case: handshake title device lba [--count N]

The header of each frame (and thus each block) holds a slot for a title-key. ↳2.6 Reading this slot for each encrypted DVD-Pack takes a rather long time. ↳9.7 Real-time playback wouldn't be possible.

Luckily, the title-key is the same for each DVD-Pack of the VMG. Also, the title-key is the same for each DVD-Pack of one and the same VTS. The title-key 00:00:00:00:00 is stored for all the areas that contain no DVD-Pack; e.g. areas for the DVD's directories, areas for IFO and BUP, and even for the gaps within a VMG or VTS.

Note: Only about 20 DVD-Videos have been checked since reading all title-keys takes a long time.

10.20 Data Padding

Test-Case: pack id decrypted-file

The prediction of a plain-text feature-data-snippet is easy if an encrypted DVD-Pack contains a padding PES (BE_H). ↳6.9

There are 5,071,051 padded DVD-Packs out of all DVD-Packs (1.07%).

There are 659,989 padded DVD-Packs out of all scrambled DVD-Packs (0.46%).

All 867 VMG+VTS have at least one padded DVD-Pack. However, only 526 out of 770 encrypted VMG+VTS have at least one padded DVD-Pack that is scrambled (68%).

10.21 Data Cycles

Test-Case: predict (cycle | libdvdcss | plus) decrypted-file

A plain-text feature-data-snippet can be predicted if the data holds a cycle *w*+.[↳6.9] The quality *Q* of the prediction is given by the number of bytes (up to the encryption border) that match the word *w*. The match *M* is the number of bytes (after the encryption border) that matches the word *w*. The table below lists for each *Q* ≥ *1* the matches *M* (%) for all scrambled DVD-Packs.

Q	Occurrence	*M=5*	*M=6..9*	*M≥10*	*Q*	Occurrence	*M=5*	*M=6..9*	*M≥10*	*Q*	Occurrence	*M=5*	*M=6..9*	*M≥10*
-	86,816,904	-	-	-	35	3,541	1.07	5.45	82.72	70	1,213	0.08	0.91	96.87
1	51,052,396	0.02	0.03	0.04	36	5,054	0.89	3.24	89.18	71	909	0.44	0.66	95.38
2	843,988	0.56	1.26	1.72	37	3,770	1.01	5.12	84.56	72	2,233	0.31	1.30	97.04
3	787,410	0.52	55.06	2.86	38	3,420	0.85	7.57	82.63	73	988	0.61	1.72	95.65
4	93,615	1.98	8.23	15.39	39	3,209	0.69	6.23	82.77	74	15,816	1.68	6.08	83.61
5	154,138	1.60	4.42	74.73	40	4,796	0.96	3.15	85.26	75	577	0.00	1.91	96.53
6	34,866	8.93	10.28	32.18	41	3,152	0.63	5.14	85.53	76	1,816	0.17	0.39	97.80
7	26,689	3.26	9.74	42.98	42	3,569	0.98	4.09	88.82	77	601	0.33	1.66	97.00
8	63,937	1.18	2.99	20.82	43	2,867	0.84	5.76	85.21	78	770	0.26	1.82	95.58
9	18,939	3.04	8.05	53.16	44	4,088	0.59	4.84	89.02	79	622	0.48	1.45	96.62
10	15,215	3.99	8.20	52.55	45	4,166	0.74	3.89	57.75	80	1,684	0.12	0.48	98.46
11	15,166	2.54	8.28	51.96	46	3,375	1.10	5.66	86.76	81	584	2.57	1.03	95.21
12	656,316	0.03	98.04	1.48	47	2,779	0.76	7.56	83.70	82	2,663	0.94	2.18	92.08
13	10,063	1.78	9.74	66.31	48	24,777	0.28	1.07	97.46	83	724	0.00	1.38	97.79
14	11,514	3.76	6.69	75.66	49	2,827	0.99	5.27	80.23	84	37,862	2.13	3.28	87.81
15	9,952	1.63	5.81	74.21	50	4,516	0.86	3.43	89.84	85	358	0.28	0.28	97.77
16	9,230	1.40	6.79	78.23	51	2,629	0.65	5.29	85.24	86	2,418	0.17	0.62	98.59
17	7,792	1.59	6.38	77.62	52	4,594	0.70	3.44	91.55	87	1,515	0.00	0.66	99.01
18	111,509	0.18	0.43	98.00	53	2,811	0.46	5.76	86.48	88	2,024	0.44	0.44	98.32
19	7,206	1.07	5.52	76.91	54	2,612	0.96	8.15	83.77	89	2,404	0.08	0.04	99.50
20	7,419	2.21	4.95	84.19	55	2,333	0.60	8.02	84.01	90	955	0.94	15.81	79.58
21	6,406	1.76	5.15	82.97	56	3,494	1.35	4.78	88.41	91	8,189	0.01	0.57	99.19
22	10,789	0.77	7.56	85.87	57	2,236	0.67	7.07	82.20	92	2,050	0.00	0.44	99.27
23	7,308	3.33	12.07	65.97	58	39,230	0.89	2.94	93.11	93	422	0.71	0.95	97.16
24	7,436	2.23	11.92	73.67	59	2,096	1.00	3.72	81.87	94	328,792	0.00	0.00	99.98
25	7,235	0.79	5.65	87.56	60	3,060	2.19	2.32	88.20	95	634	0.16	0.16	80.13
26	5,167	1.28	6.87	84.56	61	1,958	0.10	3.37	82.94	96	1,353	0.07	0.30	99.41
27	4,756	2.08	7.53	80.97	62	104,934	0.73	3.24	92.06	97	349	0.00	0.29	97.42
28	10,759	0.34	9.97	86.08	63	6,271	0.10	0.18	96.57	98	1,476	0.07	1.22	97.63
29	5,378	1.21	6.02	83.56	64	2,167	1.25	0.92	91.83	99	295	0.00	0.34	96.61
30	4,651	3.20	6.64	72.54	65	981	0.10	2.04	86.34	100	639	0.31	0.31	98.28
31	4,060	1.43	6.65	77.04	66	1,674	0.18	1.08	65.71	101	225	0.44	0.00	97.78
32	5,226	1.15	5.34	86.28	67	1,001	0.80	1.20	92.01	102	273	0.37	0.37	98.90
33	3,902	1.72	7.12	81.60	68	2,192	0.32	1.05	95.35	103	92,224	0.33	1.20	97.00
34	4,128	1.60	6.27	82.63	69	982	0.20	1.93	91.75	104	449,194	0.02	0.08	99.81

Note: if there are several cycles in a DVD-Pack, the cycle with the longest match *M* was chosen.

Example: ... x a b c d e a b c d e a b | c d e a b c d e a b c y ...

So: *w* = abcde, *h* = ab, *t* = cde, *Q* = *7* and *M* = *11*. The bar (|) illustrates the encryption border at offset 80_H.

A match *M* with 10 bytes or more will enable the recovery of a title-key. There are 1,679,529 out of 142,073,477 scrambled DVD-Packs (1.18%) with a match M ≥ 10. 70% of the matching DVD-Packs carry video data ($E0_H$) and 30% carry audio and subtitle data (BD_H).

A match *M* with 5 to 9 bytes may or may not enable the recovery of a title-key.

All but one of the 770 encrypted VMG+VTS contain at least one DVD-Pack with a match *M ≥ 10*. For the one VTS without a match (which doesn't hold the main feature of the DVD-Video) no title-key can be recovered (there are only a few matches with *M = 5*).

DeCSSplus[3] and libdvdcss[4] use variants of the cycle prediction which are a little less yielding:
– DeCSSplus appears not to be able to recover the title-key for 7 VTS (none of them main feature),
– libdvdcss appears not to be able to recover the title-key for 6 VTS (none of them main feature).

Note: the implementation used by libdvdcss appears somewhat flawed. It may happen that the prediction (which should be plain-text) holds also scrambled bytes. However, the only harm in this is wasting a little CPU time.

10.22 Title-Key Recovery

Test-Case: recover title-key file --quality Q [--retry 0|1]

The table below summarizes the result of the title-key recovery for all 145 encrypted features with 770 encrypted VMG+VTS and 97 not-encrypted VMG+VTS.

		Q=1	*Q=2*	*Q=10*	*Q=10 / Q=1*
Number of examined DVD-Packs	Total	6,325,442	6,423,847	6,804,657	6,805,633
	Not Scrambled	6,227,210	6,290,967	6,537,574	6,538,089
	Too Small	1	1	1	1
	No Prediction	60,202	130,862	265,629	265,867
	Failed Decryptions	38,029	2,017	1,453	1,676
	Title-Key recovered	769	768	766	769
	Failed	1	2	4	1
	VMG/VTS not scrambled	97	97	97	97
Duration[33] (seconds)	Total	508	225	205	212
	per DVD	3.50	1.55	1.41	1.46
	per VMG/VTS	0.59	0.26	0.24	0.24
	CPU User Time	326	39	27	29
	CPU System Time	28	28	23	22

33 On an Intel Celeron CPU G530 @ 2.40GHz. All the feature data where stored on a hard disk drive. Note: several duration values appear to vary severely for identical test cases and test data. So comparison of the results should be treated with caution.

The time to recover a title-key depends on the number of DVD-Packs to examine and the number of failed decryptions.

The higher the number of DVD-Packs to examine, the higher the required amount of data to read. This will delay the recovery process on a slow device. Besides, there are VMG/VTS which are not scrambled at all. Hence it might be an option on a slow device to examine only the first few DVD-Packs at the beginning of each VMG/VTS and stop if none of them are scrambled.[4]

A high number of failed decryptions (which are a result of a low quality prediction) will delay the recovery time on a slow CPU. However, only the quality *Q=1* will produce most patterns. As a compromise (on a slow CPU), one might start with a quality *Q=10* and retry with *Q=1* if no title-key was recovered before (see the last table column).

11 Bibliography

[1] Derek Fawcus, CSS authentication and decryption, 1999 (C-source code)

URL: *ftp://ftp.ami.com.au/pub/decss/css-auth.tar.gz*

Derek Fawcus published 1999 the re-engineered CSS C-source code and put it under GPLv2[34].

[2] Frank A. Stevenson, Cryptanalysis of Contents Scrambling System

URL: *http://www.cs.cmu.edu/~dst/DeCSS/FrankStevenson/mail1.txt*

27 Oct 1999: keystream correlation exploit↳6.6 to recover a seed from a keystream (with C-source code).

URL: *http://www.cs.cmu.edu/~dst/DeCSS/FrankStevenson/mail2.txt*

28 Oct 1999: player-key keystream exploit↳6.5 to recover the keystream from a disc-key's cipher and plain-text. Thereafter the the player-key can be constructed with the keystream correlation exploit above (with C-source code).

URL: *http://www.cs.cmu.edu/~dst/DeCSS/FrankStevenson/mail3.txt*

30 Oct 1999: disc-key exploit↳6.4 to recover a disc-key from its hash-value (with C-source code).

URL: *http://www.cs.cmu.edu/~dst/DeCSS/FrankStevenson/analysis.html*

13 Nov 1999: In this paper, Frank A. Stevenson summarizes the CSS cryptographic system and its weak spots including the three exploits above. This is the updated version of the paper from 8th November 1999.

[3] Ethan Hawke, DeCSSplus feature prediction, 2000 (C-source code)

URL: *http://www.filewatcher.com/m/DeCSSplus_v1.0.zip.33939-0.html*

Ethan Hawke takes advantage of the MPEG-2 bit-stream compression: if there is a black screen, there are a lot of repetitions. These repetitions can be exploited to predict↳6.9 a part of the feature, and so to recover the title-key.

[4] VideoLAN: libdvdcss, Version 1.2.9, 2005 (C-source code)

URL: *http://download.videolan.org/pub/libdvdcss/1.2.9/libdvdcss-1.2.9.tar.bz2*

The developers of libdvdcss put all the research above[1],[2],[3] together and made their software available on many platforms. The library is designed to access DVD-Videos without having to bother about the CSS encryption. Many open source projects use libdvdcss for playback, for instance the VLC media player and the MPlayer. The high number of installations proves the reliability of the library. So the source code has become the best available documentation on CSS.

34 http://www.gnu.org/licenses/gpl-2.0.html

[5] Jim Tailor, DVD Demystified, McGraw-Hill Companies , 2nd edition 2001

This is an invaluable secondary source of documentation, especially considering that many of the official DVD specifications are only available for a license fee, and, for the cost of a non-disclosure agreement. Note, there is a newer 3rd edition from 2005.

The author hosts also the famous DVD FAQ *http://www.dvddemystified.com/dvdfaq.html*.

[6] Standard ECMA-119, 2nd edition, 1987, Volume and File Structure of CDROM for Information Interchange

URL: *http://www.ecma-international.org/publications/standards/ecma-119.htm*

This document specifies the structure of compact read only optical disks (CD-ROM). This is the ECMA release of the ISO-9660 standard, available as a free download.

[7] Standard ECMA-267, 3rd edition, 2001, 120 mm DVD - Read-Only Disk

URL: *http://www.ecma-international.org/publications/standards/Ecma-267.htm*

The official DVD-ROM specification that defines the physical disc-layer. It is also adopted as ISO/IEC 16448.

[8] Multi-Media Commands - 5 (Working Draft) T10/1675D, Revision 4, 2006-10-24

URL: *http://www.t10.org/members/w_mmc5.htm*

The official interface specification (draft) to access a DVD-ROM drive (besides others).

[9] OSTA Univeral Disk Format Specification, 1996, revision 1.02

URL: *http://www.osta.org/specs/pdf/udf102.pdf*

The official UDF specification as a subset of ISO/IEC-13346.

Section 6.9 describes requirements for DVD-ROM and DVD-Video. It also defines the so-called UDF bridge disc which includes support for the ISO-9660[6] file-system on DVD-Video.

The source code distributed with this project does not support UDF at all, but only ISO-9660 (on such bridge discs). It appears that many DVD-Videos implement ISO-9660. ↳10.4

[10] DVD-Video Information (MPlayer)

URL: *http://dvdnav.mplayerhq.hu/dvdinfo*

URL: *git://git.videolan.org/libdvdread.git*, *git://git.videolan.org/libdvdnav.git*

DVD-Video is a proprietary video format and official format specifications are not publicly accessible. Though, there are people who provide open source software to play DVD-Video. The site shows results from re-engineering the DVD-Video format. However, the descriptions appear not to be complete. An alternative source of information, for people familiar with the C-programming language, is the source code.

[11] International Standard ISO/IEC 13818-1, 2nd edition, 2000, Information technology - Generic coding of moving pictures and associated audio information: Systems

Official specification of the MPEG-2 Transport and Program Stream (PS). DVD-Packs ↳2.9 appear to be a subset of MPEG-2 PS packs.