ETSI TS 102 825-5 V1.1.1 (2008-07)

Technical Specification

Digital Video Broadcasting (DVB); Content Protection and Copy Management (DVB-CPCM); Part 5: CPCM Security Toolbox

Reference
DTS/JTC-DVB-222-5

Keywords
broadcast, DVB

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

Contents

Intellectual Property Rights

IPRs essential or potentially essential to the present document may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (http://webapp.etsi.org/IPR/home.asp).

Pursuant to the ETSI IPR Policy, no investigation, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Foreword

This Technical Specification (TS) has been produced by Joint Technical Committee (JTC) Broadcast of the European Broadcasting Union (EBU), Comité Européen de Normalisation ELECtrotechnique (CENELEC) and the European Telecommunications Standards Institute (ETSI).

NOTE: The EBU/ETSI JTC Broadcast was established in 1990 to co-ordinate the drafting of standards in the specific field of broadcasting and related fields. Since 1995 the JTC Broadcast became a tripartite body by including in the Memorandum of Understanding also CENELEC, which is responsible for the standardization of radio and television receivers. The EBU is a professional association of broadcasting organizations whose work includes the co-ordination of its members' activities in the technical, legal, programme-making and programme-exchange domains. The EBU has active members in about 60 countries in the European broadcasting area; its headquarters is in Geneva.

European Broadcasting Union
CH-1218 GRAND SACONNEX (Geneva)
Switzerland
Tel: +41 22 717 21 11
Fax: +41 22 717 24 81

Founded in September 1993, the DVB Project is a market-led consortium of public and private sector organizations in the television industry. Its aim is to establish the framework for the introduction of MPEG-2 based digital television services. Now comprising over 200 organizations from more than 25 countries around the world, DVB fosters market-led systems, which meet the real needs, and economic circumstances, of the consumer electronics and the broadcast industry.

The present document is part 5 of a multi-part deliverable. Full details of the entire series can be found in part 1 [i.1].

Introduction

CPCM is a system for Content Protection and Copy Management of commercial digital content delivered to consumer products. CPCM manages content usage from acquisition into the CPCM system until final consumption, or export from the CPCM system, in accordance with the particular usage rules of that content. Possible sources for commercial digital content include broadcast (e.g. cable, satellite, and terrestrial), Internet-based services, packaged media, and mobile services, among others. CPCM is intended for use in protecting all types of content - audio, video and associated applications and data. CPCM specifications facilitate interoperability of such content after acquisition into CPCM by networked consumer devices for both home networking and remote access.

This first phase of the specification addresses CPCM for digital Content encoded and transported by linear transport systems in accordance with TS 101 154 [2]. A later second phase will address CPCM for Content encoded and transported by systems that are based upon Internet Protocols in accordance with TS 102 005 [1].

1 Scope

The present document specifies the Security Specification for the Digital Video Broadcasting (DVB) Content Protection and Copy Management (CPCM) system.

2 References

References are either specific (identified by date of publication and/or edition number or version number) or non-specific.

- For a specific reference, subsequent revisions do not apply.
- Non-specific reference may be made only to a complete document or a part thereof and only in the following cases:
  - if it is accepted that it will be possible to use all future changes of the referenced document for the purposes of the referring document;
  - for informative references.

Referenced documents which are not found to be publicly available in the expected location might be found at http://docbox.etsi.org/Reference.

For online referenced documents, information sufficient to identify and locate the source shall be provided. Preferably, the primary source of the referenced document should be cited, in order to ensure traceability. Furthermore, the reference should, as far as possible, remain valid for the expected life of the document. The reference shall include the method of access to the referenced document and the full network address, with the same punctuation and use of upper case and lower case letters.

NOTE: While any hyperlinks included in this clause were valid at the time of publication ETSI cannot guarantee their long term validity.

2.1 Normative references

The following referenced documents are indispensable for the application of the present document. For dated references, only the edition cited applies. For non-specific references, the latest edition of the referenced document (including any amendments) applies.

[1] ETSI TS 102 005: Guidelines for the use of compression formats over IPDigital Video Broadcasting (DVB); Specification for the use of Video and Audio Coding in DVB services delivered directly over IP protocols"Digital Video Broadcasting (DVB); Specification for the use of Video and Audio Coding in DVB services delivered directly over IP protocols".

[2] ETSI TS 101 154: "Digital Video Broadcasting (DVB); Specification for the use of Video and Audio Coding in Broadcasting Applications based on the MPEG-2 Transport Stream".

[3] FIPS Publication 180-1: "Secure Hash Standard, National Institute of Standards and Technology, 1994.

NOTE: Available at http://www.itl.nist.gov/fipspubs/fip180-1.htm.

[4] FIPS Publication 198 (2001): "The Keyed-Hash Message Authentication Code (HMAC), National Institute of Standards and Technology".

NOTE: Available at http://csrc.nist.gov/publications/fips/fips198/fips-198a.pdf.

[5] FIPS Publication 197: Advanced Encryption Standard, National Institute of Standards and Technology, 2001. .

NOTE: Available at http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

[6] FIPS Special Publication 800-38A (2001): "Recommendation for Block Cipher Modes of Operation".

NOTE: Available at http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf.

[7] ISO-IEC-13818-1: "Information technology - Generic coding of moving pictures and associated audio information: Systems".

[8] PKCS #1 (v2.1) (2002): "RSA Cryptography Standard" RSA Laboratories.

NOTE: Available at ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-1/pkcs-1v2-1.pdf.

2.2 Informative references

The following referenced documents are not essential to the use of the present document but they assist the user with regard to a particular subject area**.** For non-specific references, the latest version of the referenced document (including any amendments) applies.

[i.1] ETSI TS 102 825-1: "Digital Video Broadcasting (DVB); Content Protection and Copy Management (DVB-CPCM); Part 1: CPCM Abbreviations, Definitions and Terms".

[i.2] ETSI TS 102 825-4: "Digital Video Broadcasting (DVB); Content Protection and Copy Management (DVB-CPCM); Part 4: CPCM System Specification".

[i.3] ETSI TS 102 825-3: "Digital Video Broadcasting (DVB); Content Protection and Copy Management (DVB-CPCM); Part 3: CPCM Usage State Information".

3 Definitions, abbreviations and notation

3.1 Definitions

For the purposes of the present document, the terms and definitions given in TS 102 825-1 [i.1] apply.

3.2 Abbreviations

For the purposes of the present document, the abbreviations given in TS 102 825-1 [i.1] apply.

3.3 Notation

The present document uses the following notation throughout:

Table 1: Notation

Scope	Notation	Meaning
Hashes	H(M)	The Hash of message M.
	H(X,Y)	The Hash of the concatenation of messages X and Y, X being first.
MAC	MAC{K}(M)	The MAC of message M using key K.
Encryption	E{K}(M)	Encryption of message M using key K.
Decryption	D{K}(C)	Decryption of cipher text C using key K.
LSA	XOR, ⊕ or ^	Bitwise XOR operation on two blocks of 16 bytes.
	AND	Bitwise AND.
	Padding	The LSA works on blocks of 16 bytes. If an input block contains less then 16 bytes it is padded with zeroes at the high end, i.e. the highest byte(s) of the resulting basic block will be zero, before further usage. A box containing two zeroes on the right side of a partial block indicates this.
	||	Concatenation of two partial blocks to form a full block of 16 bytes, e.g. C_n||00.
	MSC_x	The x^{th} MSC data block of 16 bytes, or possibly less for the last block.
	P_x	The x^{th} plaintext block of 16 bytes, or possibly less for the last block.
	C_x	The x^{th} ciphertext block of 16 bytes, or possibly less for the last block.
	00^a	A string of *a* bytes with value 0.
General Values	Abs(x)	Absolute value of x.
	Array	All Array indices start at 0.
	Array[*a*]	Indexing: the $(a + 1)^{th}$ element in an array.
	i = a…*b*	This represents a range of numbers from *i* = *a* to *i* = *b* inclusive, $a,b \in Z$.

4 Cryptographic Algorithms

4.1 Hash function

CPCM compliant implementations shall use the SHA-1 hash algorithm described in [3].

4.2 Message Authentication Code

CPCM compliant implementations shall use the HMAC Message Authentication Code generation and verification algorithm described in [4].

NOTE: Message Authentication Code are sometimes also named signature when the key is AD Secret or SAC Session key.

4.3 Symmetric cipher

CPCM compliant implementations shall use the symmetric cipher algorithm Advanced Encryption Standard (AES) with 128-bit key length as described in [5].

For SAC protection, cipher is used in Cipher Block Chaining (CBC) mode as described in [6] with a fixed Initialization Vector (IV) value for both License Protection and as a Secure Authenticated Channel (SAC) cipher. If message length is not dividable by 16, the last message block shall be padded with sufficient bytes of value 0x00 to get a full block. Upon decryption, padding shall be removed using the message length.

For protection of Content License with AD Secret, cipher is used in Electronic Code Book (ECB).

4.4 Revocation Lists Digital Signature

Revocation Lists are signed using RSASSA-PKCS1-V1_5-SIGN described in PKCS #1 v2.1 [8]. CPCM compliant implementations do not need to implement RSASSA-PKCS1-V1_5-SIGN. Revocation Lists signatures are verified using RSASSA-PKCS1-V1_5-VERIFY described in PKCS #1 v2.1, [8]. The public exponent used in the signature process is constant (see clause 6.3).

4.5 Local Scrambler Algorithm (LSA)

The LSA described in this clause shall be used to protect Content within compliant CPCM systems. The LSA uses the AES cipher described in [5] as its basic building block, with 128-bit keys, called Control Words (CWs).

The basic unit of scrambling (or descrambling) is called a packet. Each packet is scrambled independently from all others, allowing random access. A packet consists of a part that must not be scrambled, called the Must Stay Clear (MSC) data, (aka MSC Data or MSC Part), and a part that must be scrambled, called the payload. The MSC Part consists of 0 or more bytes. The LSA has two so-called *MSC modes* for handling the MSC data:

- "MSC Data Dependent" (MDD): MSC Data contributes to the scrambling of the rest of the packet. A change of the data in the MSC Part (PID, PCR, etc. contained in the header or Adaptation Field) of the scrambled packet will adversely impact de-scrambling. The MSC Part may contain bits that must be masked, i.e. set to 0, prior to processing of the MSC Part in this mode, for instance an error bit which may be set in transit from a scrambling source to a descrambling sink. Processing of the MSC part in this mode is explained later in this clause.

- "MSC Data Independent" (MDI): the MSC Data does not contribute to the scrambling of the rest of the packet. A change of the data in the MSC part (e.g. for re-multiplexing) may be done without descrambling and re-scrambling.

A third possibility is to encrypt/decrypt the complete packet, i.e. including the MSC data. This is controlled by the *MSC Override* signal.

The LSA uses AES encryption/decryption on 16-byte blocks. Part of the result of processing one block is used for processing the next block. This process is called chaining. The LSA supports two *chaining modes*: CBC and RCBC. Depending on the particular adaptation it may be possible that the payload within a packet that is to be scrambled is not a multiple of 16 bytes and is greater than 16 bytes. This is true for the MPEG-2 TS adaptation defined in CPCM phase 1. In such a case a Ciphertext Stealing (CS) technique is used to process the last full 16-byte block and following partial block in the packet.

If less than 16 bytes needs to be scrambled/descrambled a Short Block Handling technique is used.

Figure 1 below depicts a black-box view of the cipher, showing the different inputs and processes. The Initial Vector (IV) used by the LSA depends on the various signals and can depend on the MSC data and/or its length, depending on the used modes.

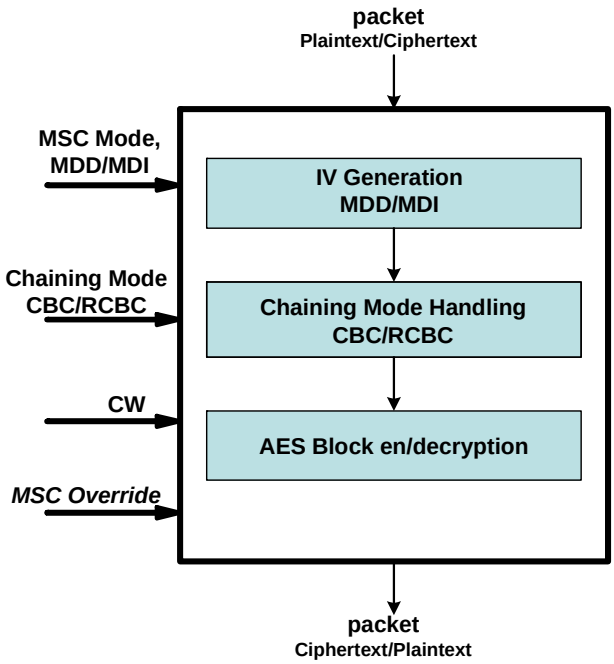


Figure 1: Scrambler/descrambler - block diagram

Clauses 4.5.1 through 4.5.4 describe the generic part of the LSA informatively. Clause 4.5.5 gives a normative description of this generic LSA specification. Clause 4.5.6 describes the LSA MPEG-2 Transport Stream adaptation normatively.

4.5.1 Control signals and parameters

Figure 2 shows the scrambling/descrambling process on a very high level. The length of the MSC Part is denoted as *MSCL*. Table 2: lists the LSA parameters and control signals.

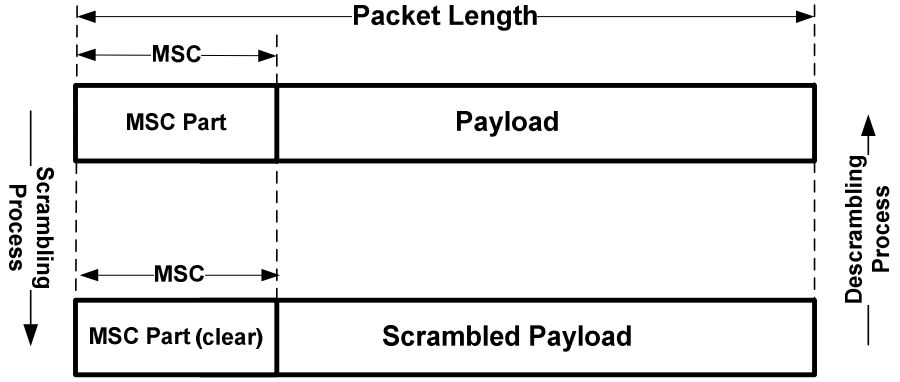


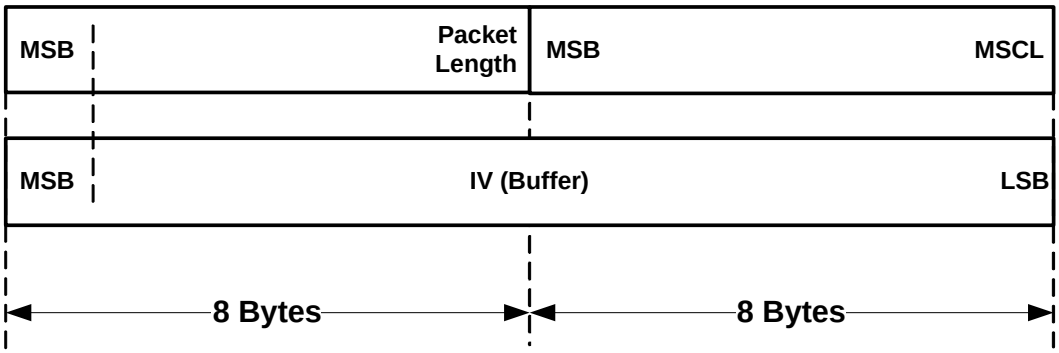
Figure 2: High-level view of the scrambling/descrambling process

Table 2: Parameters and control signals

Parameter	Description
Packet Length	The Packet Length (in bytes) shall be known to the scrambler/descrambler for initialization of the scrambling/descrambling process.
Must Stay Clear Length (MSCL)	MSCL is the number of bytes at the beginning of the packet that must stay clear, i.e. are not LSA-scrambled.
CW	128-bit Content encryption/decryption key for the AES block cipher.
Signal	**Description**
MSC Mode	Is either MDI or MDD, (MSC Mode is set to 0 or 1, respectively).
Chaining Mode	Is either CBC or RCBC (Chaining Mode is set to 0 or 1, respectively).
MSC Override	If True (set to 1), the MSCL is set to 0, (i.e. the whole packet is scrambled or descrambled.); else MSCL is derived from the packet format.

The LSA also uses the following values and notation:

Table 3: General LSA values and notation

Value/Notation	Meaning
Buffer	A 16-byte internal buffer used by the LSA engine.
IVE	IVE is the initial value used in the scrambling of the first payload block. Clause 4.5.1.1 describes how it is derived, depending on the MSC Mode used.
IVseed	IVseed is the seed for generating IVE. IVseed = f(MSCL, Packet Length), for CM = CBC; else IVseed = IVseedConstant.
IVseedConstant	Fixed 16-byte constant, defined in clause 6.3.
f(MSC, Packet Length)	f(MSCL, Packet Length) is a 16 byte IVseed, composed of the Packet Length in the most significant 64 bits of the IVseed, and MSCL in the least significant 64 bits of the IVseed. Big-endian is used, such that the 8th and the 16th byte of the IVseed are LSB of the packet length and MSCL, respectively. The process is shown in the diagram below.  **Figure 3: Forming the IVseed in CBC mode**

4.5.1.1 IV for encryption payload preparation and Short Solitary Block Handling

Figure 4 shows the preparation process for the initial value used in the scrambling of the first payload block, i.e. the Effective IV (IVE), on the left side, and the Short Solitary Block Handing (SSBH) process, on the right side. These processes are independent of the Chaining Mode, CBC or RCBC, although the value of IVseed depends on this mode.

The IVE depends on the MSC Mode. If the MDI mode is used then the IVE is just the encryption of IVseed. If the MDD mode is used then a MAC is calculated over the MSC Data. If the last MSC block contains less then 16 bytes then it is padded with zeroes on the least significant side.

The *Mask* process masks the bits in the packet header that must not be used to calculate the MAC.

If MSC Override is True, there is no MSC Data, (MSCL = 0), and the IVE is derived as if the MSC Mode is MDI. The encryption key is the same as used for the encryption or decryption of the payload, i.e. the Control Word.

If a packet contains a payload of less then 16 bytes (but at least one), these are (de-)scrambled by XOR-ing them with the most significant bytes of the IVE. This is called SSBH.

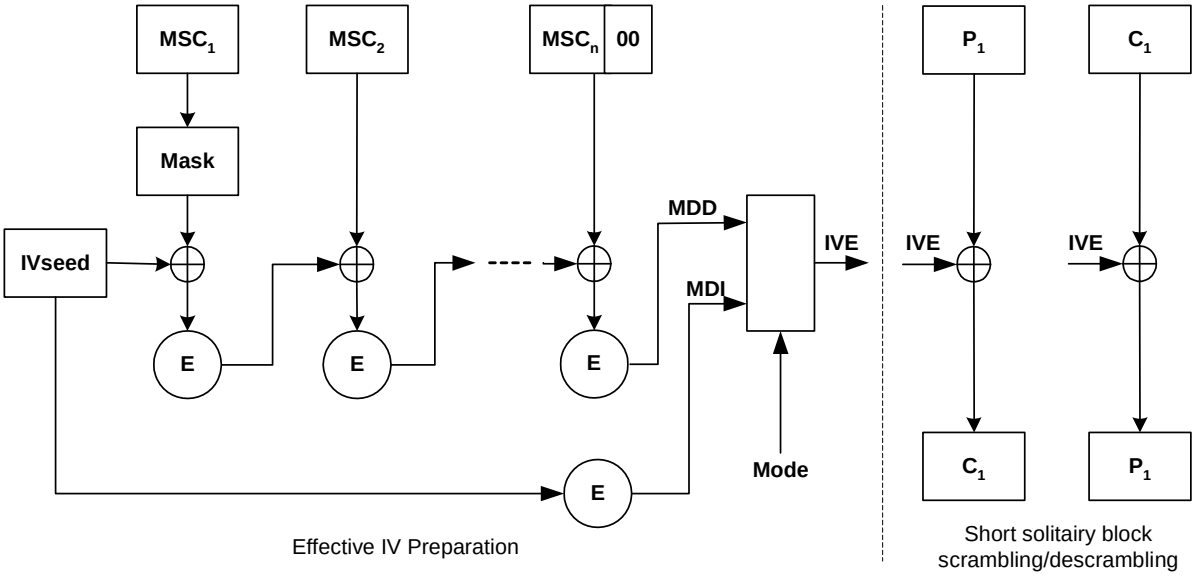


Figure 4: IVE preparation and Short Solitary Block Handling (SSBH)

4.5.2 CBC Chaining Mode

Figure 5 shows the CBC encryption (top part) and decryption (bottom part) of *n* plaintext blocks $P_1 \ldots P_n$ and ciphertext blocks $C_1 \ldots C_n$. The left side shows the case that the non-MSC Data in a packet constitutes an exact number of basic blocks (i.e. is a multiple of 16 bytes). The right side shows the Ciphertext Stealing (CS) process used when the size *s* of the last block, P_n and C_n, is less than a basic block.

In both cases scrambling starts by XOR-ing the first plaintext block P_1 with the IVE and encrypting the result to yield the first ciphertext block C_1, which is used instead of the IVE for the encryption of the next block, and so on, until the last plaintext block is encrypted. If CS is applied, then the last plaintext block is padded with *s* zeroes at the least significant side to get a block of 16 bytes. In addition the two last ciphertext blocks are swapped and the least significant *s* bytes of the now last block (C' in the figure) are cut off.

Descrambling starts with the decryption of the first ciphertext block, C_1. The result is XOR-ed with the IVE to yield the first plaintext block, P_1. In the next stage C_1 is used instead of IVE, and so on. If the payload is a multiple of 16 bytes this ends with the last block, i.e. $P_n = D(C_n) \oplus C_{n-1}$. If CS is used, then this process stops with $P_{n-2} = D(C_{n-2}) \oplus C_{n-3}$ (for $n>2$). The result of decrypting the next block, C_{n-1}, is XOR-ed with the padded last ciphertext block, $C_n \parallel 00$, to yield the *last*, partial plaintext block, P_n, concatenated with the recovered ciphertext part, C', which was discarded after the scrambling phase. Next, the padding of C_n is replaced with this recovered part and the result, $C_n \| C'$, is decrypted and XOR-ed with C_{n-2} to yield P_{n-1}. Finally P_n and P_{n-1} are swapped and the padding of P_n is discarded.

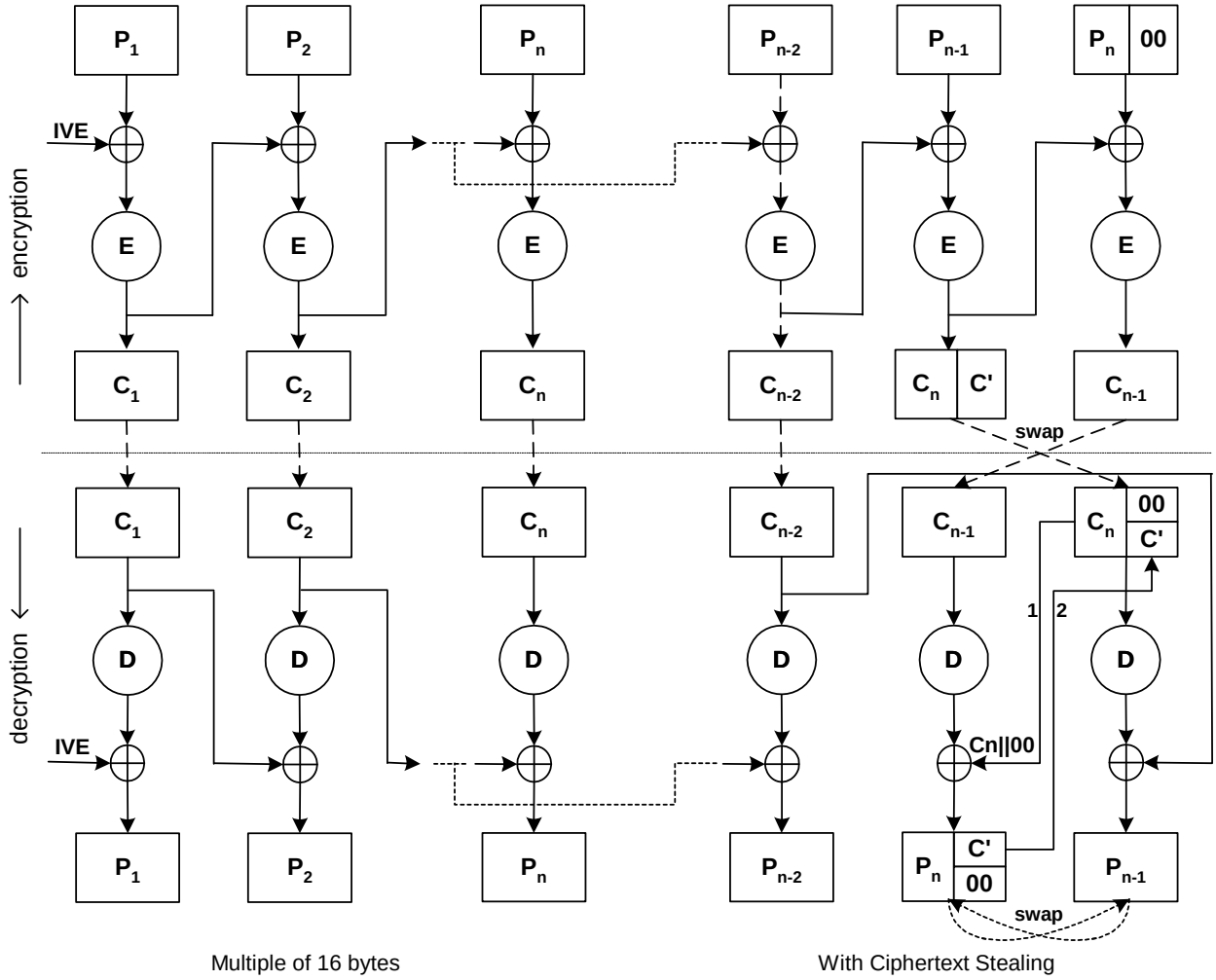


Figure 5: CBC encryption and decryption, with and without Ciphertext Stealing (CS)

4.5.3 RCBC Chaining Mode

Figure 6 shows the RCBC encryption (top part) and decryption (bottom part) of *n* plaintext blocks $P_1 \ldots P_n$ and ciphertext blocks $C_1 \ldots C_n$, organised as *n/2* super blocks. The part on the right shows the Ciphertext Stealing (CS), which is only used if the size of the last block, P_n and C_n, is less than a basic block.

The encryption of each complete super block starts with XOR-ing IV_n with the last plaintext basic block of the super block. The result is encrypted to yield the last ciphertext block of the super block. The encryption of the XOR of this block and the previous plaintext block yields the previous ciphertext block, and so on. All super blocks are processed in normal order.

If CS is applied to the last super block it starts with the XOR of the *first* plaintext block, P_{n-1}, with $IV_{n-1} \wedge P_n$, i.e. IVE XOR-ed with all plaintext blocks except P_{n-1}. The result is encrypted to yield the *last* ciphertext block, which is divided into two parts, C' of size 16-*s* ($0 < s < 16$) and C_n of size *s*. The whole block is XOR-ed with the last, *s*-byte plaintext block P_n padded with zeroes, the encryption of which yields C_{n-1}. The ciphertext blocks are output from the scrambler in the order C_1 to C_n, with C' omitted.

The order of processing blocks in a super block is not reversed for descrambling, which starts with the decryption of the first ciphertext block, C_1, of the first super block. The result is XOR-ed with the next ciphertext block, C_2, to yield the first plaintext block. C_2 is XOR-ed with IV_2 to yield P_2. This is repeated for all super blocks except for the last one if CS is applied. IV_n is calculated exactly as during encryption, e.g. by XOR-ing IVE with all previously recovered plaintext blocks (and optionally with the MSC Data). If CS is applied descrambling of the last super block starts when C_{n-1} and the *s*-byte block C_n are received and the latter is left-padded with zeroes. First, C_{n-1} is decrypted and the result is XOR-ed with the zero-padded C_n to yield P_n left-padded with C', which equals the omitted part in the scrambler. Next, the zero-pad of C_n and C'-pad of P_n are exchanged and the now C'-padded C_n block is decrypted to yield P_{n-1}. The plaintext blocks are swapped so the descrambler output is in the right order again, i.e. from P_1 to P_n.

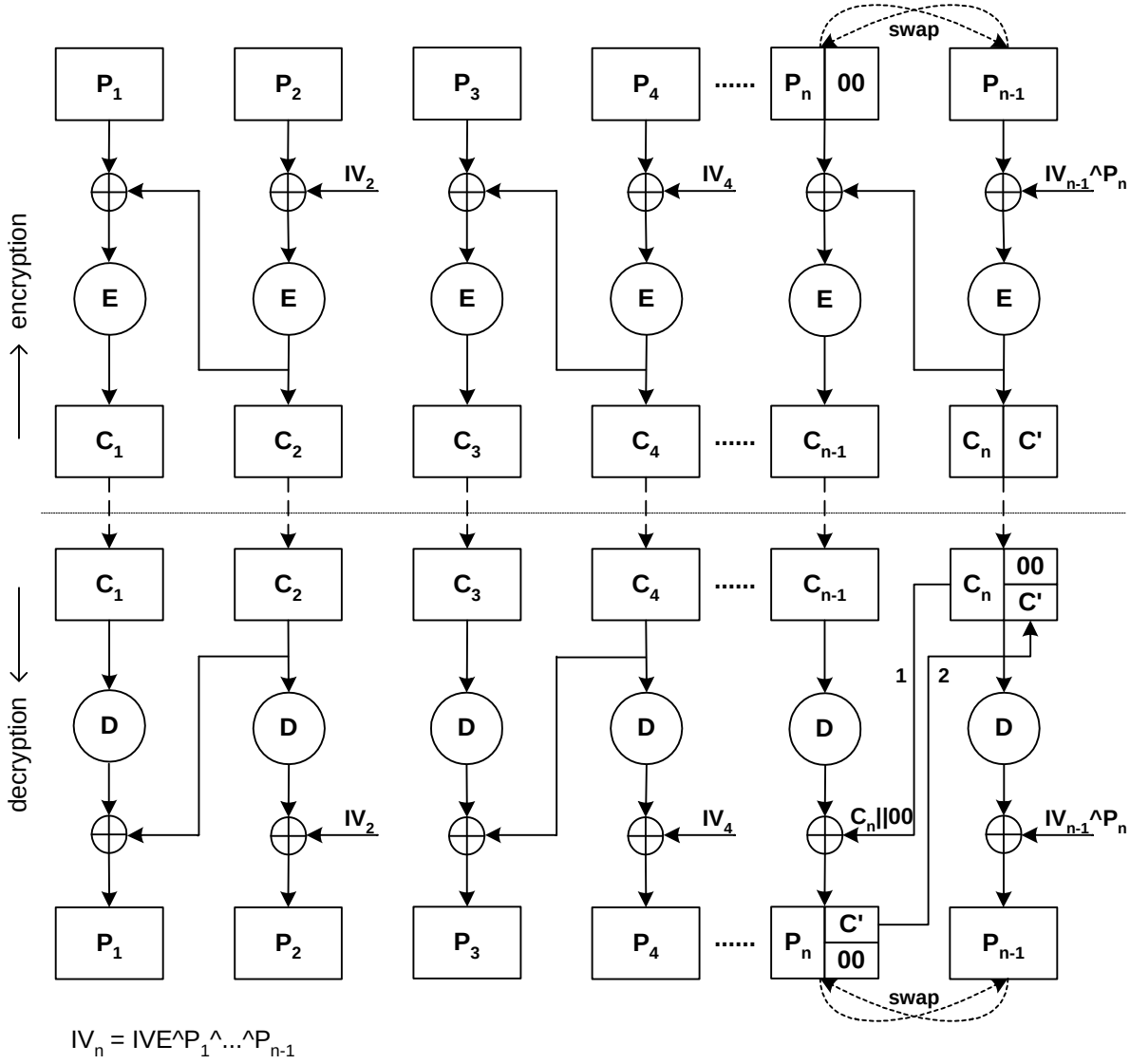


Figure 6: RCBC encryption and decryption, with ciphertext Stealing (CS)

Figure 6 above shows the case where the data that is to be (de-)scrambled consists of an even number of basic blocks, each of which are 16 bytes long, except for the last one. The first *n*-2 blocks are processed as super blocks, while the last full and partial block are processed using CS. If all blocks would have been full all would have been processed as super blocks. If the number of blocks is odd and more then 1, the block before the last two blocks, i.e. P_{n-2} or C_{n-2}, is processed as if it is the last block of a super block.

EXAMPLE: If n = 5, P_1 and P_2 would be encrypted as a super block, P_3 would first be XOR-ed with IV_3 (= IVE $\oplus$ P_1 $\oplus$ P_2) and then encrypted to yield C_3, and P_4 and P_5 would either be handled as a super block or using CS when P_5 is a partial block.

Clause 4.5.5 provides a normative description of the scrambling/descrambling process, covering all cases.

4.5.4 Summary of CBC and RCBC scrambling/descrambling process

If at least 16 bytes need to be scrambled/descrambled this is done using either CBC or RCBC, possibly with Ciphertext Stealing. The process can be summarized as follows.

Table 4:Scrambling modes

Mode	Action	Do if *nr* bytes left	bytes left afterwards	Next action
CBC	CBC	$(nr = n)$ or $(nr \geq 2n)$	0 or $\geq 2n$	CBC or CS
	CS	$n < nr < 2n$	0	none
RCBC	SB	$(nr = 2n)$ or $(nr > 3n)$	0 or $> n$	SB or 1B or CS
	1B	$(nr = n)$ or $(2n < nr \leq 3n)$	0 or $(n < nr \leq 2n)$	none or SB or CS
	CS	$n < nr < 2n$	0	none

The complete process can be described concisely as:

- CBC mode: CBC*,CS?
- RCBC mode: SB*,1B?,(SB|CS)?

The following definitions apply:

Table 5: Scrambling mode definitions

	Do		Mode	Description
*	Do zero or more times		CBC	Encrypt/decrypt 1 basic block using Cipher Block Chaining
?	Do zero or one time		CS	Do Ciphertext Stealing
a|b	Do either a or b		1B	Encrypt/decrypt 1 basic block with the current IV_n
a,b	Do a first, then b		SB	RCBC encrypt/decrypt a Super Block of 2 basic blocks

4.5.5 LSA - Normative Specification

This clause provides the formal description of the LSA. Inputs, outputs, arrays and addition are as defined in clauses 3.1, 3.2, 3.3 and 4.1 in [4]. IVseed is a described in Table 3.

If MSC Override is True or MSC Mode is MDI, the Effective IV (IVE) is:

IVE = E{CW}(IVseed)

Else, (MSC Mode is MDD and MSC Override is not True), IVE is calculated as follows:

NOTE: *MSCL* = (16*q* + *r*), with *q*, *r* ∈ Z. $B = b_0\, b_2 \ldots b_{15}$, is a 16-byte buffer. The *Masking* of certain bits in the MSC is not shown, because it depends on the particular adaptation.

$B = MSC_1 \,||\, 00^{16-r}$, q = 0

$B = MSC_1$, q > 0

B = E{CW}(B ⊕ IVseed)

$B = E\{CW\}(MSC_i \oplus B)$, $i = 2 \ldots q$

if $r > 0$: $B = E\{CW\}(MSC_{q+1} \,||\, 00^p)$

IVE = B

The number of bytes to encrypt or decrypt is $b = 16q + r$ = (packet length – MSCL) ≥ 1, and $p = 16 – r$, with $p, q, r \in$ Z. For instance, if MSC Override is False, the packet length is 188 and MSCL = 4 then $b = 184$, $q = 11$, $r = 8$, $p = 8$,

Short Solitary Block Handling (SSBH, $b < 16$) is independent of the Chaining Mode:

$C_1[i] = IVE[i] \oplus P_1[i]$, $i = 0…b\text{-}1$

$P_1[i] = IVE[i] \oplus C_1[i]$, $i = 0…b\text{-}1$

If $b = 16$ scrambling and descrambling is the same for both Chaining Modes, although the result is different because of a different IVseed and therefore a different IVE:

$C_1 = E\{CW\}(P_1 \oplus IVE)$, $b = 16$

$P_1 = \{CW\}D(C_1) \oplus IVE$, $b = 16$

If $b > 16$, the scrambling/descrambling depends on the Chaining Mode.

If the Chaining Mode is CBC, process all blocks using CBC if $b = 16c$, $c = q$, or full blocks until one full and one partial block is left if $b = 16c + d$, $c = q – 1$, $16 < d < 32$.

$C_1 = E\{CW\}(P_1 \oplus IVE)$, $C_i = E\{CW\}(P_i \oplus C_{i-1})$, $i = 2…c$.

$P_1 = D\{CW\}(C_1) \oplus IVE$, $P_i = D\{CW\}(C_i) \oplus C_{i-1}$, $i = 2…c$.

Apply Ciphertext Stealing if $16 < d < 32$:

$C_2 \parallel C'_p = E\{CW\}(P_1 \oplus IVE)$, $C_1 = E\{CW\}((P_2 \parallel 00^p) \oplus (C_2 \parallel C'^p))$, $c = 0$.

$P_2 \parallel C'_p = D\{CW\}(C_1) \oplus (C_2 \parallel 00^p)$, $P_1 = D\{CW\}((C_2 \parallel C'^p) \oplus IVE$, $c = 0$.

$C_{c+2} \parallel C'^p = E\{CW\}(P_{c+1} \oplus C_c)$, $C_{c+1} = E\{CW\}((P_{c+2} \parallel 00^p) \oplus (C_{c+2} \parallel C'^p))$, $c > 0$.

$P_{c+2} \parallel C'^p = D\{CW\}(C_{c+1}) \oplus (C_{c+2} \parallel 00^p)$, $P_{c+1} = D\{CW\}((C_{c+2} \parallel C'^p) \oplus C_c$, $c > 0$.

If the Chaining Mode is RCBC:

$IV_i = IVE \oplus P_1 \oplus…\oplus P_{i-1}$ (E.g. $IV_1 = IVE$, $IV_2 = IVE \oplus P_1$, $IV_3 = IVE \oplus P_1 \oplus P_2$, …)

While there are still bytes to encrypt or decrypt do one of the 3 actions shown below::

1) If more then 3 or exactly 2 full blocks are left, process the (next) 2 blocks as a super block:

   $C_{i+1} = E\{CW\}(P_{i+1} \oplus IV_{i+1})$, $C_i = E\{CW\}(P_i \oplus C_{i+1})$, $i \in [1, 3, …]$

   $P_i = D\{CW\}(C_i) \oplus C_{i+1,}$ $P_{i+1} = D\{CW\}(C_{i+1}) \oplus IV_{i+1}$, , $i \in [1, 3, …]$

2) If exactly 1 full block is left, or 2 full blocks and one partial or full block, process the next block as if it is the last of a super block, i.e. $b = 32c + d$, ($d = 16$) or ($32 < d \leq 48$):

   $C_{2c+1} = E\{CW\}(P_{2c+1} \oplus IV_{2c+1})$

   $P_{2c+1} = D\{CW\}(C_{2c+1}) \oplus IV_{2c+1}$

3) If more than 1 but less than 2 blocks are left use Ciphertext Stealing, i.e. if $b = 16c + d$, $16 < d < 32$, $p = 32–d$:

   $C_{c+2} \parallel C'^p = E\{CW\}(P_{c+1} \oplus IV_{c+1} \oplus P_{c+2})$, $C_{c+1} = E\{CW\}((P_{c+2} \parallel 00^p) \oplus (C_{c+2} \parallel C'^p))$

   $P_{c+2} \parallel C'^p = D\{CW\}(C_{c+1}) \oplus (C_{c+2} \parallel 00^p)$, $P_{c+1} = D\{CW\}(C_{c+2} \parallel C'^p) \oplus IV_{c+1} \oplus P_{c+2}$

4.5.6 MPEG-2 Transport Stream adaptation

This clause describes the adaptation of the LSA to scramble/descramble MPEG-2 Transport Streams as described in [1]. The unit of scrambling is a 188-byte MPEG-2 Transport Stream (TS) packet.

The MSC Part of a packet consists of the header and the (optional) Adaptation Field (AF) bytes. In MDD mode: The *transport error indicator* bit in the TS header is *Masked*, i.e. it is set to zero before it is used to calculate the MAC. This bit is located in the second byte of MSC_1 (see [7]). This is normatively described by inserting the next line after the "B = MSC_1 , $q > 0$" line in clause 4.5.5: "B [2] = B[1] AND {0x7F} (This masks the transport stream error bit.)".

For MPEG-2 TS packets MSC Override is 0.

4.6 Certificate verification

Certificate signature verification shall be performed using RSA with recovery and fixed exponent *e*.

The Certificate signature shall be verified using the following steps:

1) Identify the public RSA signature key used to sign the certificate and the corresponding certificate, henceforth designated as the parent certificate. If the parent certificate is not the root certificate (see clause 5.2), the CPCM Instance must first verify:

   1) that the parent Certificate is valid (using the same steps);

   2) that the parent Certificate is not revoked; and

   3) that the parent Certificate has the is_signer field set to 1.

2) Apply the RSA public exponent to the certificate: $m = s^e \bmod n$; possibly pad the result with leading zeroes so that its length is the same as the RSA public modulus.

3) Break the result *m*, into 16 blocks of 128 bits each: $M_1, \ldots, M_{16}$, with M_1 the most significant block. Verify that:

   $M_1 \equiv \text{Digest}(IV_{Certificate}, M_2, M_3, \ldots, M_{16}) \text{ (modulo } 2^{127})$,

   i.e. compare the payload digest to M_1 only in the 127 LSBs, ignoring the MSB, where:

   - IVCertificate is a non-secret universal CPCM constant that is part of the CPCM standard.
   - Digest is defined as follows:
     - $\text{Digest}(X_1) = X_1$
     - $\text{Digest}(X_1, X_2, \ldots, X_{N+1}) = E\{\text{Digest}(X_1, X_2, \ldots, X_N)\}(X_{N+1}) \oplus X_{N+1}$, for $N \geq 1$

   Each X_i being a 128-bit block.

   $M_2||M_3\ldots||M_8$ constitute the Certificate fields other than the compressed modulus or Diffie-Hellman public key (see clause 6.1).

   $M_9||M_{10}\ldots||M_{16}$ constitute the Diffie-Helman public key if `is_signer` is not asserted.

   $M_9||M_{10}\ldots||M_{16}$ constitute the compressed modulus if `is_signer` is asserted.

   Before this modulus may be used (as the public key for the RSA signature verification of a "child" Certificate, or as the public key for RSA encryption of a challenge value in a challenge-response handshake protocol between two CPCM Instances), the CPCM security kernel shall expand (decompress) it, as described below.

4) Verify whether the issuer_id certificate field matches the identifier of the parent certificate.

5) Verify whether the Certificate is trusted based on the C&R regime mask certificate field (see clause 5.2).

6) Verify that the certificate field is_revocation is not set.

7) Verify that the Certificate is not revoked (see clause 4.8).

If any of the above steps fails, Certificate is viewed as not valid. If the Certificate or one of its parent Certificates is revoked, the result of the verification is revoked and the output revocation index is the highest of all output indices.

Certificate expiration verification shall not be part of the Certificate Verification Process. It is only used for Content Management purposes (see TS 102 825-4 [i.2]).

The CPCM security kernel shall take the following steps to expand (decompress) this compressed modulus:

1) Identify the *CPCM Certificate Id* from the appropriate field in the standard body fields clause of the recovered Certificate.

2) Let S_0 denote the *CPCM Certificate Id*, right-padded with zero-bits so that S_0 is thus 128 bits.

3) For $i = 1 ... 8$,

   Let $S_i = E\{C\}(S_{i-1})$

   where C is a non-secret universal CPCM constant defined in Table 10.

4) Form the expanded (decompressed) modulus by concatenating S_i and last half of M blocks alternately, starting with S_i (in the most significant bits of the expanded modulus) and ending with M_{16} (in the least significant bits), i.e. using the sequence in Table 6:

Table 6: Decompressed modulus

S_1	M_9	S_2	M_{10}	…	S_8	M_{16}

5) If the most significant bit of the expanded modulus (i.e. the first bit of S_1) is 0, set it to 1 in order to guarantee that the modulus is "full size".

4.7 Certificate keys and digest generation

This clause is provided as reference for Certificate Providers. CPCM Instances need not implement the described algorithm.

CPCM Certificate keys depends on the type of the certificate. A CPCM Signing Certificate contains a compressed RSA modulus, whereas a leaf Certificate contains a public Diffie-Hellman key. CPCM Certificates are signed by the private RSA key of their signing Certificate. The associated public RSA key of the signer is stored as a compressed modulus in the signed Certificate. The steps for creating a compressed modulus for signing Certificates shall be as follows:

1) Let S_0 denote the *CPCM Certificate Id*, right-padded with zero-bits so that S_0 is thus 128 bits.

2) Let $S_i = E\{C\}(S_{i-1})$, for $i = 1…8$, where C is a non-secret universal CPCM constant defined in Table 10 in clause 6.3.

3) Force the most significant bit (i.e. the first bit) of S_1 to 1.

4) Choose a 1 000-bit prime p such that $(p-1)$ is not divisible by 65 537.

5) Build the following 10-dimensional lattice generated by the following 9 vectors x_i ($i = 0…8$):
   $x_{i,i+1} = 2^{2048}$ for $i = 0…6$
   $x_{i,j} = 0$ for $i = 0…6$, $j \neq i+1$
   $x_{7,j} = 2^{256j}p$ modulo 2^{2048} for $j = 0…7$
   $x_{7,8}=1$
   $x_{7,9}=0$
   $x_{8,j}=2^{1920}(S_{j+1} + 0.5)$ for $j = 0…7$
   $x_{8,8}=0$
   $x_{8,9}=2^{1920}$
   where $x_{i,j}$ means the j-th coordinate of the i-th vector.

6) Find an alternative vector base y_i (i = 0…8) that generates the same lattice, consisting of vector with small norms, using, for example, the LLL algorithm.

7) If $y_{i,9} \neq 0$ for exactly one value of i, change if necessary the order of the vectors, so that after the change of order $y_{8,9} \neq 0$; Otherwise, restart from step 2.

   Search for a prime number q such that (q – 1) is not divisible by 65 537 amongst values:

   $$\text{Abs}(y_{8,8} + \sum_{i=0}^{7} c_i y_{i,8}) \text{ where } |c_i| \leq 2 \text{ for } i = 0\ldots7;$$ if not found, restart from step 2.

8) Break pq into 16 128-bit blocks N_1 (most significant), N_2, N_3, …, N_{16} and check whether $N_{2i+1}=S_{i+1}$ for i=0…7. If yes, pq is the required modulus, and N_2, N_4,…, N_{16} form its compressed representation to be inserted as the least significant half (M_9,M_{10},…,M_{16}) into a Certificate. If not, restart from step 2.

The other parts of the Certificate are a hash value and a body part (see Table 7). The hash, which is the first 16-byte block of the Certificate, is calculated over the remainder of the Certificate as the Digest function described in point 6 in clause 4.6, but with the MSB always set to 0.

The Certificate is signed using the private RSA key of the signing Certificate.

4.8 Revocation List verification

When a new Content Revocation List is received, a CPCM Instance shall first verify that list and if this succeeds proceed to check the revocation status. If the verification fails, the new Revocation List shall be ignored and discarded.

The CPCM Instance shall verify all new Revocation Lists as follows:

1) If the received version is lower than its stored one, verification fails.

2) If the received version is equal to, and the received index is lower than or equal to its stored one, then verification fails.

3) Finally check the signature of the received list in the following way:

   a) Verify the Certificate of the C&R regime that issued the Revocation List using the Certificate verification procedure described in 4.6, and verify that the `is_revocation` bit in this Certificate is set to 1. Upon successful verification, the public modulus of the C&R regime is obtained.

   b) Verify the signature of the Revocation List itself using the algorithm referred to in clause 4.4.

4) If check 3 is successful, verification succeeds. Else, it fails.

Once a CPCM compliant Instance has verified the Revocation List it will then check the revocation status as follows:

1) If the 'to be verified' CPCM Certificate Identifier Generation Index is lower than the one indicated in the list, then the Certificate is revoked and the output reference revocation index is set to the index of the Revocation List when the Generation Index last changed.

2) Else, if the CPCM Certificate Identifier is included in the Revocation List, then the Certificate is revoked and the value of the First Revocation Index associated with the Certificate in the Revocation List is returned.

3) Else, the Certificate is not revoked.

To check the revocation status of its AD, a CPCM compliant Instance shall first have a valid Revocation List. Then, it proceeds as follows:

1) It computes the digest of its AD Secret (ADS) using the Hash function defined in clause 4.1.

2) If the ADS digest is included in the Revocation List, then the AD is revoked and the value of the First Revocation Index associated with the ADS in the Revocation List is returned.

3) Else, the AD is not revoked.

When a Certificate or an AD is revoked, a reference revocation index shall be output. This reference revocation index is used in the Content License verification and ignored in other cases (i.e. the Certificate is considered as revoked).

4.9 Secure time management

A CPCM Instance may be able to securely determine both Secure Absolute Time and Secure Relative Time as described in the following:

1) Secure Absolute Time. This measures CPCM time as defined in the System Specification (TS 102 825-3 [i.3]). The way to implement such a Secure Absolute Time source is out of the scope of the present document and will be subject to the C&R regime. The implementation of Secure Absolute Time source is optional.

2) Secure Relative Time. This measures the number of seconds since a given event (e.g. Acquisition of a given piece of Content, establishment of a SAC, contact with a Secure Absolute Time source, and so on). The implementation of a Secure Relative Time source is mandatory and shall at least be able to securely measure SAC session key expiry time (please see clause 5.1).

CPCM Instances shall continuously maintain secure time whenever they are active.

A CPCM Instance that is not Secure Absolute Time capable on its own may obtain it from another CPCM Instance using the following procedure:

1) It first discovers Instances that are Secure Absolute Time capable (see above).

2) If such a CPCM Instance is present, it shall establish a SAC with that Instance. If it already has a SAC with that CPCM Instance, it shall renew the SAC. During the SAC establishment or renewal protocol, it shall verify that the peer CPCM Instance Certificate has the field `absolute_time_capable` set. If not, it shall restart the process at point 1 with another CPCM Instance if one exists.

3) Once the SAC has been established, it may request the current secure absolute time using the *get_absolute_time()* message sent in the clear.

4) The peer CPCM Instance shall answer using the message *CPCM_absolute_time()*, containing a field that represents the Secure Absolute Time as defined above. This message shall be authenticated using the SAC authentication key.

5 Cryptographic Protocols

5.1 Authenticated Key Exchange (AKE)

DVB CPCM compliant implementations shall use the following protocol to establish trust and create a SAC. The protocol is based on Diffie-Hellman. Each CPCM entity shall have a random private key (K_{priv}) and a Certificate that embeds an identity (e.g. the Certificate serial number) and the public key ($K_{pub}=g^{Kpriv}$ mod p where g and p are Diffie-Hellman parameters shared by all entities). For the sake of simplicity, the notation mod p will be omitted in the rest of the document but it shall be understood that all exponentiations of g are made modulo p.

The following is the sketch protocol between Alice and Bob.

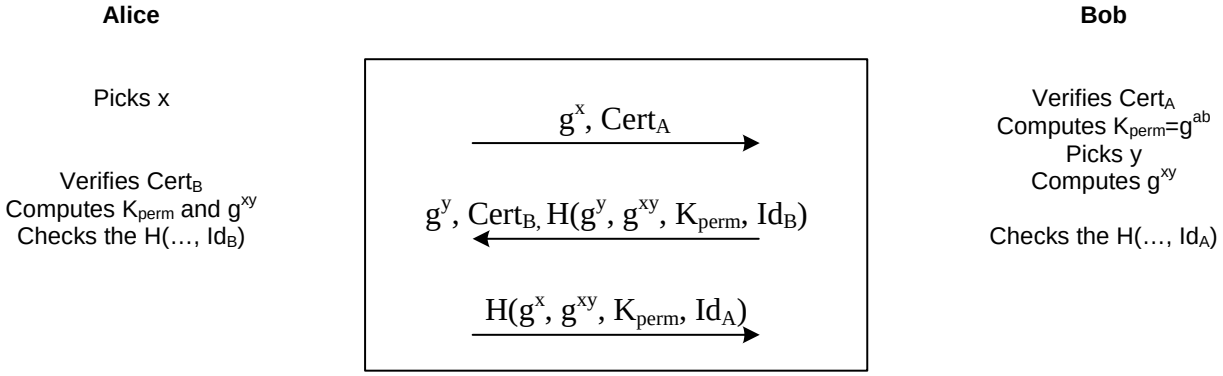


Figure 7: Authenticated Key Exchange

1) Alice picks a random x, computes the associated public value g^x and sends the result to Bob together with her Certificate $Cert_A$ in the call AKE_init(). In the case of session key renewal the Certificate Identifier may be sent in place of $Cert_A$.

2) Bob extracts the public key g^a and the identity Id_A. He verifies that the Certificate is valid (see clause 4.6). He then computes the secret key $K_{perm}=g^{ab}$.

3) Bob picks a random y, and computes the associated public value g^y. He also computes the hash value of the concatenation of g^y, g^{xy}, K_{perm} and Id_B. He sends the result of both computations together with his Certificate to Alice in the call AKE_Commit(). In the case of session key renewal the Certificate Identifier may be sent in place of $Cert_B$.

4) Alice extracts the public key g^b and the identity Id_B. She verifies that the Certificate is valid (see clause 4.6). She then computes the secret key $K_{perm}=g^{ab}$.

5) Alice computes g^{xy} and checks that the received hash value is correct. She then computes the hash value of the concatenation of g^x, g^{xy}, Kperm and Id_A and sends the result to Bob in the call AKE_Confirm().

6) Bob verifies the correctness of received hash value.

7) Both CPCM Instances compute the SAC symmetric session keys K_{sess_enc} and K_{sess_auth}:

$$K_{sess\_enc} = H(00||g^{xy}, K_{perm})_{[0..16]}$$

$$K_{sess\_auth} = H(01||g^{xy}, K_{perm})_{[0..16]}$$

K_{sess_enc} is the encryption authentication session key.

K_{sess_auth} is the SAC authentication session key.

NOTE 1: Where the notation $X_{[0..y]}$ means the first y bytes of message X.

If one or both Certificates are revoked, AKE aborts unless AKE was performed for Content Management purposes. In such case, behaviour described in TS 102 825-4 [i.2] shall be adopted.

NOTE 2: Step 7 may be omitted if only trust establishment is needed, i.e. no SAC establishment.

SAC session keys expire after SAC session key expiry time (see TS 102 825-4 [i.2] for definition). SAC session keys renewal shall be renewed once expired. This key renewal can occur as well before the expiration to ensure SAC session keys are always available.

The mechanism for synchronizing the SAC session keys that are in use is described in the CPCM System Specification (TS 102 825-4 [i.2]).

5.2 Trust Management

Each CPCM Instance has CPCM credentials in the form of a Certificate chain that provides the Instances public key and other attributes. Trust establishment between different CPCM Instances shall be based on Certificate chain exchange and verification. The Certificate chain has a tree hierarchy with a single source of trust, namely the Root Authority, a chain of (RSA) signing Certificates, and a terminating Instance Certificate that holds a Diffie-Hellman public key, which is used for the trust establishment protocols described above. Signing Certificates are issued by Certificate Authorities or their descendants.

Root Authority certificates shall be stored in each CPCM Instances in write-protected memory. Root certificates include CPCM version, certificate identifier and RSA public key. As root certificates are never exchanged, no specific format is given for the root certificate storage and no signature is required.

The structure and management of such Certificate chains are described below.

The CPCM Trust Management hierarchy starts at the Root Authority, which shall be able to certify any number of Certificate Authorities designated by one or more of the DVB CPCM C&R regimes, as depicted in Figure 8 below; there will be no more than eight C&R regimes.

These Certificate Authorities may issue Signing Certificates to other entities wishing to produce Signing Certificates or Instance Certificates subject to the C&R regimes constraints. Thus commercial entities may choose to generate their own Certificates.

EXAMPLE: A device manufacturer may structure its Certificate hierarchy according to their multiple business units and their product lines; the dotted line in Figure 8 indicates such an unspecified hierarchy.

In contrast, other commercial entities have the option to get their Instance Certificates from an outside Certificate Authority rather than generate their own Instance Certificate chain in-house (shown by a direct line on the same figure).

All Certificates, with the exception of the Instance (or leaf) Certificates, are Signing Certificates containing the RSA public key needed in the verification of its descendants. Signing Certificates will also indicate for which C&R regime or regimes they are valid by means of the setting of a bit or bits in an 8-bit C&R mask field preserved and expressed in the Instance Certificate.

Each C&R regime is responsible for defining its own compliance and robustness rules and revocation policy, and issuing Revocation Lists.

The Revocation Lists shall be generated by the C&R regimes and implicitly signed (protected) by the Root Authority.

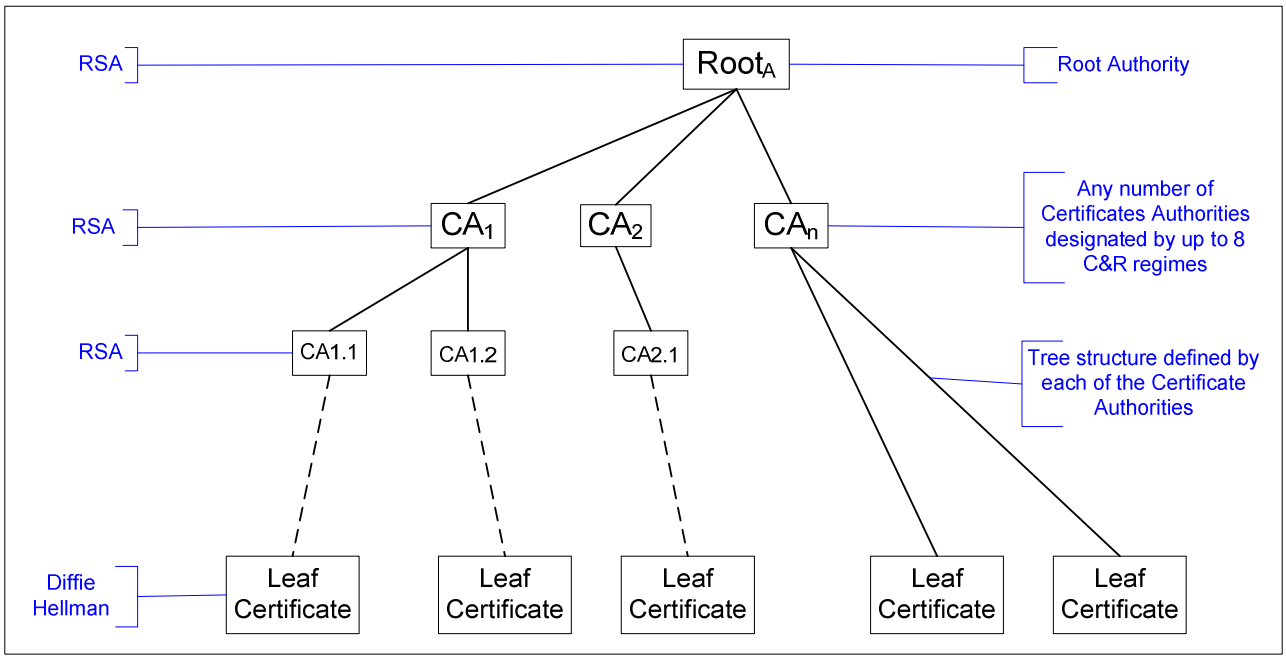


Figure 8: Certificate hierarchy

The common Root Authority enables two CPCM Instances not implementing the same C&R regime to trust each other.

A CPCM Instance A will trust another CPCM Instance B if the following conditions are met:

- At least one of the C&R regimes that A conforms to and for which B is not revoked trusts at least one of the C&R regimes of B for the intended action (e.g. ADM action or Proximity check).
- The Certificate of B and each Revocation Lists of the C&R regimes that A conforms to are verified.

Trust relationship is not reflective. A may trust B without B trusting A.

NOTE: There are several reasons for using such a hierarchy where the Root is not a Certificate Authority and does not provide Instance Certificates:

- **Security enhancement**
  Limiting the number of times a Root private key must be used for signing greatly reduces the probability of it leaking out.
- **Structured efficient revocation or Certificate attributes update**
  If many of a particular Certificate's "descendants" are compromised or need renewal, the burden of individual revocation or renewal of each descendent may be replaced by the revocation or renewal of the common ancestor. This may also be applied to individual field updates.
- **Commercial flexibility and lower cost**
  To allow CPCM licensees to be their own Certificate Authority and generate Certificates for their Instances without having to be tethered to an external Certificate Authority.

6 Formats and definitions

6.1 Certificate format

Table 7 shows the CPCM Instance Certificate after verification as defined in clause 4.6. Table 8 shows the structure of the certificate body, which corresponds to the Certificate field blocks, $M_2,\ldots,M_8$, that are recovered as part of the Certificate verification process. This structure is described in detail in TS 102 825-4 [i.2]. Here only the aggregation rules for the fields are described. Aggregation means that the value of the field in the Certificate is superseded by the value of the same field of another Certificate in the chain. For example, a Certificate lower down in the chain (i.e. further from the root) may not able to grant capabilities or permissions broader than its ancestor's capabilities and permissions.

All Certificate fields for which aggregation applies use either the *Minimum* value of the field across all the Certificates in the chain or the *Maximum* value. For fields without an aggregation rule only the value present in the Certificate shall be applied. The meaning of the aggregate rule is summarized in Table 9. In more details, the Certificate chain for a CPCM Instance comprises a set of Signing Certificates (where the is_signer field is set to 1) terminated by an Instance Certificate (where the is_signer field is set to 0). The chain has to be verified and the aggregation rules are applied per Certificate field resulting in a potential update to the Instance Certificate's fields to become the "actual" field value.

Table 7: Unsigned CPCM Instance Certificate

Syntax	Bits	Identifier
unsigned_cpcm__instance_certificate () {		
certificate_hash	128	bslbf
certificate_body_fields	896	bslbf
compressed_public_key	1024	bslbf
}		

Table 8 : Standard Body Fields of a Certificate

Syntax	Aggregate Rule
certificate_body_fields () {	
CPCM_version = 0x01	N/A
instance_ID	N/A
CPCM_instance_certificate_ID	N/A
issuer_ID	N/A
C_and_R_regime_mask	Minimum
certificate_expiration_time	Maximum
generation_index	N/A
is_signer	N/A
is_revocation	N/A
reserved	N/A
content_handling_capability	N/A
AD_aware	Minimum
ADM_capable	Minimum
ADM_LM_capable	Minimum
ADM_DC_capable	Minimum
ADSE_countable	N/A
LSA_capable	N/A
absolute_time_capable	Minimum
geographic_aware	N/A
reserved_for_extensions	
}	

Table 9: Aggregate rule meanings

Rule	Meaning
N/A	Do not apply an aggregation rule; use the value in the Certificate.
Maximum	Use the maximum value in the chain as the actual value in the Instance Certificate.
Minimum	Use the minimum value in the chain as the actual value in the Instance Certificate.

6.2 Revocation List hierarchy

The revocation list format can be found in TS 102 825-4 [i.2]. CPCM Compliant devices shall support a simple two level root and branch Revocation List hierarchy as in Figure 9.

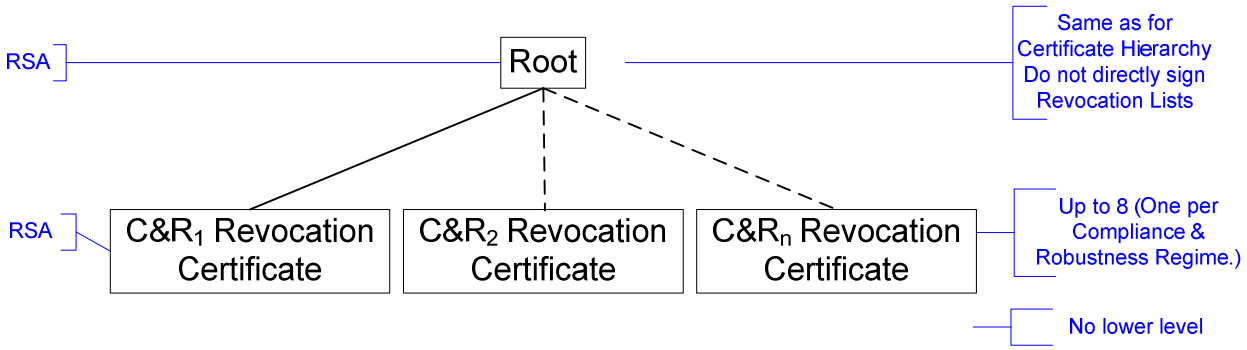


Figure 9: Revocation List Hierarchy

6.3 Cryptography Toolbox parameters

CPCM compliant implementations shall use the following parameters.

Table 10: Cryptography Toolbox parameters

Context	Parameter	Value
Certification	Public Exponent	$e=2^{16}+1=65537$
	RSA modulus size	2048 bits
	$IV_{certificate}$	C0 AC 29 B7 C9 7C 50 DD 3F 84 D5 B5 B5 47 09 17
	C	24 3F 6A 88 85 A3 08 D3 13 19 8A 2E 03 70 73 44
SAC	Group generator	g=2
	Modulus, *p*	*p* is a strong prime of 1024 bits, represented below using Big-endian convention. DA B6 B0 94 B2 C5 6A 0E D1 6B C4 6E F6 04 CF D9 BA 34 04 CA C4 BF 65 96 49 97 D0 DD C6 C5 A0 D0 75 9F AF C4 67 44 45 74 57 57 8B CC 3C 70 7B F7 C2 6A 3B A9 DF A5 CD 27 D2 E1 9F 60 DF D3 37 D0 A0 51 EC CC 3B 82 4B 63 09 D6 FC 5C DB 7E E0 41 EA 56 32 78 CB 05 4C 1B 54 25 0A C1 FB 00 D8 91 15 22 DC F6 38 C3 02 75 B3 82 46 14 69 69 35 39 FB 89 E9 FC EC 47 5A 1A F2 FD D3 9C BF B0 C8 DB
License and SAC protection	Fixed IV value	44 56 42 20 54 4D 2D 43 50 54 20 43 50 43 4D 2E
Scrambling for MPEG-2 Transport Streams	Packet Length	188 bytes
	MSC	4 + optional Adaptation Fields [3].
	Mask	All "1"s except for the most significant bit in byte 1 (bytes are numbered from 0, so it is in fact the second byte!) which is "0".
	IVseed Constant	CB CE CB CD CB CE CB CD CB CE CB CD CB CE CB CD
Revocation	Public Exponent	$e=2^{16}+1=65537$

List of tables

List of figures

History

Document history		
V1.1.1	July 2008	Publication