

Institutionen för systemteknik

Department of Electrical Engineering

Examensarbete

Replacement of the DVB-CSA

Subtitle goes here!

Examensarbete utfört i Subject goes here
vid Tekniska högskolan vid Linköpings universitet
av

Gustaf Bengtz

LiTH-ISY-EX--YY/NNNN--SE

Linköping 2014



Linköpings universitet
TEKNISKA HÖGSKOLAN

Genomgång av nya och alternativa krypterings- och scramblingssystem för digital-teve samt implementering av ny scrambling-algoritm (DVB-CSA3)

Subtitle goes here!

Examensarbete utfört i Subject goes here
vid Tekniska högskolan vid Linköpings universitet
av


Gustaf Bengtz

LiTH-ISY-EX--YY/NNNN--SE

Handledare: **Oscar Gustafsson**
ISY, Linköpings universitet
Patrik Lantto
WISI NORDEN

Examinator: **Kent Palmkvist**
ISY, Linköpings universitet

Linköping, 21 februari 2014

		Avdelning, Institution Division, Department ISY Embedded systems Department of Electrical Engineering SE-581 83 Linköping		Datum Date 2014-02-21	
Språk Language <input type="checkbox"/> Svenska/Swedish <input checked="" type="checkbox"/> Engelska/English <input type="checkbox"/> _____		Rapporttyp Report category <input type="checkbox"/> Licentiatavhandling <input checked="" type="checkbox"/> Examensarbete <input type="checkbox"/> C-uppsats <input type="checkbox"/> D-uppsats <input type="checkbox"/> Övrig rapport <input type="checkbox"/> _____		ISBN _____ ISRN LiTH-ISY-EX--YY/NNNN--SE Serietitel och serienummer ISSN Title of series, numbering _____	
URL för elektronisk version					
Titel Title Genomgång av nya och alternativa krypterings- och scramblingsystem för digital-teve samt implementering av ny scrambling-algoritm (DVB-CSA3) Replacement of the DVB-CSA Författare Author Gustaf Bengtz					
Sammanfattning Abstract <p>This report addresses why an implementation of CSA3 is needed as a replacement to the currently used scrambling standard CSA for cryptography for IPTV data streams.</p> <p>It addresses the strengths and weaknesses of using CSA, as well as the need for encryption and scrambling of data in the DVB world.</p>					
Nyckelord Keywords problem, solving, DVB, scrambling, CISSA, cipher, CSA, CSA3					

Abstract

Här skriver jag texten som ska in i engelska abstracten. För närvarande:

Nothing to say mon.

Notation

ABBREVIATIONS

Abbreviation	Meaning
AES	Advanced Encryption Standard
CAM	Conditional Access Module
CAS	Conditional Access System
CBC mode	Cipher block chaining mode
Ciphertext	Encrypted plaintext
CISSA	Common IPTV Software-oriented Scrambling Algorithm
CPU	Central Processing Unit
CSA	Common Scrambling Algorithm
CTR mode	Counter mode
CW	Control Word
DVB	Digital Video Broadcasting
ECM	Entitlement Control Message. CW encrypted by the CAS.
EMM	Entitlement Management Messages
ES	Elementary stream
ETSI	European Telecommunications Standards Institute
IPTV	Internet Protocol Television
IV	Initialization vector
LFSR	Linear Feedback Shift-Register
LSB	Least Significant Bit
MSB	Most Significant Bit
Nibble	Half a byte (4 bits)
Nonce	A value that is only used once
P-Box	Permutation-Box
PES	Packetized Elementary Stream
Plaintext	Data
PS	Program Stream
S-Box	Substitution-Box
STB	Set-top Box
TS	Transport Stream
XRC	eXtended emulation Resistant Cipher

Contents

Notation	v
1 DVB Transmissions	1
1.1 Set-up	1
1.2 Conditional Access System	2
1.2.1 Standards	2
1.3 DVB-SimulCrypt	2
1.4 Common Interface	3
1.5 Conditional Access Module	3
2 Scrambling	5
2.1 What is the need for cryptography?	5
2.2 Scrambling or Encrypting	6
2.3 Data packets	6
2.3.1 TS packets	6
2.3.2 PES packets	7
2.4 Encryption and Decryption	7
2.4.1 Symmetric-key encryption	8
2.4.2 Public-key encryption	8
2.4.3 Combination	8
2.5 Ciphers	8
2.5.1 Block cipher	9
2.5.2 Stream cipher	10
2.6 Confusion and Diffusion	10
2.6.1 S-boxes	10
2.6.2 P-Boxes	10
2.7 Secrecy	11
3 CSA	13
3.1 Why do we need a new standard?	13
3.2 Layout of the CSA	14
3.3 Security	14
3.3.1 Brute force	15

4	CISSA and CSA3	17
4.1	CISSA	17
4.1.1	Software friendly	18
4.2	CSA3	18
4.2.1	Hardware friendly	18
4.3	Conclusion	18
5	Advanced Encryption standard	21
5.1	Method	21
5.1.1	InitialRound	22
5.1.2	SubBytes	22
5.1.3	ShiftRows	22
5.1.4	MixColumns	22
5.1.5	AddRoundKey	23
5.2	KeyExpansion	23
5.2.1	Key-schedule core	23
5.2.2	Rijndael's S-Box	23
5.2.3	Rcon	23
6	Result	27
6.1	Hardware	27
6.2	Flow	27
6.3	Special solutions	27
6.4	How much of the CSA3 standard has been implemented?	28
A	Matrixes	31
A.1	Calculations on the CBC-mode	31
	Bibliography	35

1

DVB Transmissions

There are many parts that are needed to provide DVB with a secure way of transmitting streams without facing the risk of content getting stolen. The most important parts are:

- Scrambler - explained in chapter 2
- CAS - explained in section 1.2
- Common Interface - explained in section 1.4
- Descrambler - the inverse of a scrambler.

1.1 Set-up

The *Transport Stream* (TS) is scrambled using a key which is called a *control word* (CW). While CW is at least changed every 120th second, it is not uncommon to change it as often as every 10th seconds. Because of the frequency in which you change the CW, finding the key has very little effect. Even if someone manages to find a CW, they will only be able to use it for a few seconds before the CW is changed. The control word is generated randomly to make sure that consecutive control words are not related to each other.

The control word is scrambled, then sent to the content provider. The content provider then encrypts the CW as an *entitlement control message* (ECM). The content provider also generates an *entitlement management message* (EMM) which tells the smart-card if the user is allowed access to the data. The ECM and EMM are then sent back to the scrambler where it is attached to the scrambled TS. This package is then sent to the user, where the ECM, EMM and TS are separated. The

ECM and EMM are sent to the box containing the smart-card for processing, and the TS is sent directly to the descrambler where it waits for the CW. The TS is then descrambled using the CW and the pure data is then displayed to the user.

1.2 Conditional Access System

Conditional Access (CA) is used to make sure that a user fulfills an amount of criteria before being able to view content. A CA system (CAS) consist of an EMM-generator and an ECM-generator in addition to some other parts. We are only interested in the ECMG and EMMG since they are the parts that affect the scrambler. The ECM is generated using the CW, while the EMM is generated based on information related to the user. That information varies, but might relate to what channels the user is allowed to access and when the subscription ends. A TV will not broadcast any channels without receiving an EMM telling it to do so.

A simple example is that a user needs to pay for TV-services to be able to access content. The content provider generates an EMM which tells the smart-card whether the user is allowed to access the requested material or not. The content provider also generates an ECM based on the CW, which the smart-card decrypts and passes to the descrambler if the EMM allows it.

1.2.1 Standards

Some of the CA systems currently in use are Viaccess, Conax, Irdeto, NDS, Strong and NagraVision. There are different *Conditional Access Modules* (CAM) that are used, and which one is needed depends on the content provider. NDS is used by Viasat, Conax is used by Com Hem, Viaccess is used by Boxer and Strong is used by Canal Digital for instance.

Viaccess

Viaccess is one of the most common CA systems. In Sweden it is used by Boxer and SVT.

Conax

Conax is the CA-module used by Com Hem. Conax has got its headquarter in Norway and is a part of the Telenor Group, which deals with mobile telecommunications [Conax, 2014].

Strong

Strong is the CA-module used by Canal Digital. This is currently the third most used content provider in Sweden.

1.3 DVB-SimulCrypt

DVB-SimulCrypt is widespread in Europe, and works as an interface between the head-end and the CAS [ETSI TS, 2008].

Source:
You need to find more sources than just Patrik

Remove?:
Or keep?

SimulCrypt encourages the use of several CAS at once [ETSI TS, 2008, p. 17].

This is done by sending the same CW to many CAS at the same time, and then allowing them to generate an ECM and EMM based on the CW. The multiplexer in the head-end then creates TS packets based on those, since the EMMs will determine whether the user is allowed access or not.

1.4 Common Interface

The Common Interface is the interface between the CAM and the host (Digital TV receiver-decoder). There are currently two versions of common interfaces in use, which are the CI and the CI+. The difference between them is that the output from the CI is unencrypted, while the output from the CI+ is encrypted [LLP, 2011]. This means that a clear TS is sent between the CI and the host, that can be copied. The data sent between the CI+ and host can not be copied due to it being encrypted, and therefore provides more security for content providers [European Standard, 1997].

1.5 Conditional Access Module

CA modules (CAMs) are responsible of decoding the scrambled TS received from the host. The CAM is often input to a PCMCIA slot (Personal Computer Memory International Association) either in the TV or the STB. The CAM consists of a slot for a smart-card and a descrambler. The smart card decodes the ECM and sends the CW to the descrambler. The TS is then descrambled and the clear TS is sent back to the host from the CAM.

2

Scrambling

Security is not only about cryptography. But there is a main reason why cryptography is attacked, and that is because there is a very low chance of being detected. There will be no traces of the attack, since the attacker's access will look just like a legitimate / "good" access.

This can be compared to a real-life break-in. The break-in will be noticed if the thief breaks in using a crowbar. If the thief, on the other hand, would be to pick the lock, there is a possibility that you will never notice that your security has been breached [Schneier and Fergusson, 2003].

There are many rules concerning cryptography, as there are to all sciences. The one I found most noteworthy was that one is to always assume that someone is "out to get you". Because of this, Schneier and Fergusson [2003, pp. 12–14] say that we always need to look for possible ways to break systems, to make sure that we are safe.

2.1 What is the need for cryptography?

If we communicate without encrypting the data we are sending, someone else will most likely be eavesdropping. For most people this isn't a problem, but in some instances sending secure messages can be extremely important. One example is communication during war, where a single piece of intelligence might turn the tide of the entire war. Another example is sending manuscripts, and drafts of books over the internet. These are some of the reasons as to why we want to encode messages.

Another reason for scrambling is to reduce the number of adjacent data-bits with



Figure 2.1: General layout of a data packet [ETSI TS , 2013]

the same value, as in long strings of zeros or ones.

2.2 Scrambling or Encrypting

The difference between scrambling and encryption is that scrambling is the way we generate the secret control word (key) which is used when we make the data unreadable to outsiders while encryption is the way we protect the control word during transmission.

Todo:
Det här måste jag bli säkrare på

2.3 Data packets

The data processed by the DVB systems is sent in data packets. All of them are created from ES packets (elementary streams) which generally is the output from an audio or video encoder. The ES packets are then packeted into PS, TS or PES packets and then distributed. There are three different ways of packing data, but only two of them are commonly used. The types are called TS, PS and PES. PS packets are used while data is being stored, while TS and PES are used when data is being transmitted. The ones interesting when working with DVB are therefore the TS packets as well as the PES packets.

2.3.1 TS packets

TS packets are the ones used by the DVB society, possibly due to their fixed lengths. TS packets have got a length of 188 bytes with a 4 byte long header, meaning the payload consists of 184 bytes. The layout of a TS packet can be viewed in figure 2.1.

The building blocks that the TS packets consist of are:

- Header
- Adaptation field - might not exist
- Encrypted payload - might not exist
- Clear payload

The header consist of information regarding the packet. The header provides information as to whether there is an adaptation field in the packet, what id the packet has, the sync byte and two bits telling us whether the data is scrambled

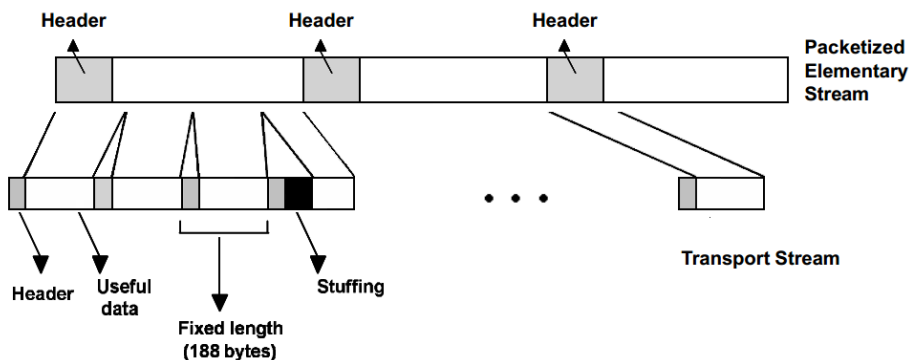


Figure 2.2: PES packet derived from TS packets (inspiration for the image taken from [ETSI, 1996, p. 9])

using an odd or even key, if it is scrambled. The header is never to be encrypted and is always found in the beginning of the packets.

The adaptation field is a padding that you input when the end of the data is not aligned with the end of the TS packet. This is done to make sure that the TS packet is filled. We only find adaptation fields we are working with the last string of data, if the data does not align. Adaptation fields are not encrypted.

We are prone to end up with clear bytes of data in the TS packets when we work with block ciphers, since block ciphers only encrypt fixed sizes of data. The clear data is always located at the end of the packets and can in a worst case scenario be up to one byte smaller than the block size, since that will be the largest amount of bytes that does not fill an entire block. The encrypted payload is always located in front of the clear payload [ETSI TS , 2013, pp. 10–11].

2.3.2 PES packets

The PES packets have varying lengths of up to 64 KBytes.

PES-packets can be of any size less than 64 Kbytes. They are often packed into TS packets when distributed, due to the strength of the TS packets. The payload data in the TS packets consists of the entire PES packets, which consists of the header and the data. PES packets do not use adaptation fields, since they can be of more or less any length, as long as the packet do not exceed 64 KBytes.

2.4 Encryption and Decryption

There are two things that you need when you encrypt and decrypt messages. Those are the algorithm as well as the key. There are plenty of ways to encrypt messages, but there are two ways of sharing the encryption-key. The first method

being the symmetric-key encryption, and the second method being the public-key encryption.

2.4.1 Symmetric-key encryption

The symmetric-key encryption uses the same key to encode and decode messages. Distribution of the key, when using the symmetric-key encryption is troublesome and the fact that both parties need access to the same secret key is a major drawback of the symmetric key encryption, as compared to the public-key encryption method. Sending the key in an email is a bad idea, since the persons who wants to read our messages most likely already will be listening, and they will therefore obtain the key as well as the means to decode the messages we send.

2.4.2 Public-key encryption

The public-key encryption uses a public key that anyone can look up, and a secret key that only one person knows [Simmons, 1992, pp. 25–32]. For instance say that the two persons, Bob and Alice, want to communicate. Bob produces a keypair P_{Bob} (Bob's public key) and S_{Bob} (Bob's secret key) and publishes P_{Bob} for anyone to see. When Alice wants to send Bob a message, she looks up Bob's public key P_{Bob} , which she then uses to encode her message. When she sends Bob the message, Bob decodes the message using his secret key S_{Bob} [Schneier and Fergusson, 2003].

2.4.3 Combination

The big question now is why we would use anything other than the public-key encryption, since it seems secure and easy to manage. The reason is that the public-key encryption is not as effective as the symmetric-key encryption. It is common to use a combination of those two since an easy and effective way to encrypt messages is what we desire. To do this we use a symmetric-key algorithm to encode the plaintext into ciphertext, and then we use the public-key encryption to encode the symmetric-key we used when encoding the plaintext. This encoded key is then sent together with the ciphertext to the recipient, who uses the secret key to decode the symmetric key, which is then used to decipher ciphertext and obtain the plaintext.

Decryption is often performed by reversing the encryption. You need to know the algorithm, preferably through a mathematical representation, to calculate how to obtain the plaintext from the ciphertext. A description of how this is done for the CBC-mode (described in 2.5.1) is described in A.1 in appendix A. We assume that we know the decryption algorithm here for simplicity.

Source:
feels like a given,
but still

2.5 Ciphers

A cipher is the same as an algorithm, that operates on either plaintexts or ciphertexts to perform encryption or decryption. Figure 2.3 describes how they can be split into smaller groups.

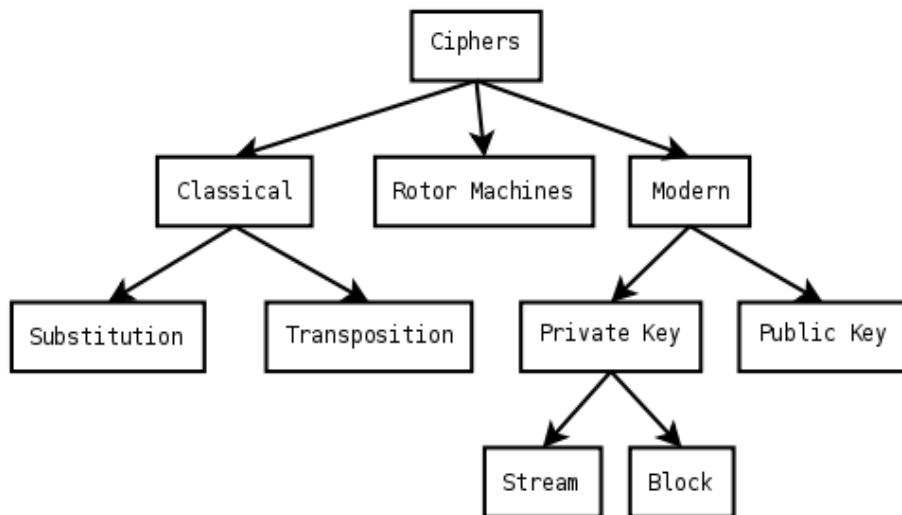


Figure 2.3: Different kinds of ciphers [Wikipedia, 2014b]

There are mainly two kinds of ciphers that are used when designing modern cryptosystems. Those ciphers are called block ciphers and stream ciphers. Many systems use a combination of block ciphers and stream ciphers to provide security.

Source:
At least the CS
uses it. But you
need a source if you
want this to be here

2.5.1 Block cipher

A block cipher operates on blocks where each block consists of a fixed number of bytes. This might cause a need for padding the blocks, in case the plaintext contains a number of bytes that is not even with the blocksize. Block cipher often use itself of a combination of S-boxes and P-boxes in a so-called SP-network (Figure 2.4).

There are many modes of block ciphers, but the two recommended by Schneier and Fergusson [2003] are the CBC-mode and the CTR-mode.

CBC stands for *cipher block chaining* and is performed by first encrypting the result of an XOR between an IV and the plaintext. This is the ciphertext that corresponds to the first plaintext. This is then put into an XOR with the next plaintext, and then encrypted [Stinson, 2006, pp. 109–111]. For reference, see image A.6 in appendix A.

CTR stands for *counter*, and refers to the way the IV is generated. The counter outputs a value, which is encoded with the key. The output is then run in an XOR together with the plaintext, producing the ciphertext. The counter is then incremented and the procedure is iterated [Stinson, 2006, p. 111].

2.5.2 Stream cipher

Stream ciphers work on a stream of data (as implied by the name). They usually consist of some kind of a keystream generator which performs a modulo 2 addition with the data [Simmons, 1992, pp. 67]. An effective implementation of the stream cipher is to use a linear feedback shift-register which uses the current internal state (key) to produce the next state by a simple XOR-addition between two or more of the bits in the state. This is mainly used because of how easy it is to construct in hardware [Fischer, 2008].

2.6 Confusion and Diffusion

Two properties that are needed to ensure that a cipher provides security are confusion and diffusion [Shannon, 1949]. Note that a cipher is not secure just because these two properties are obtained.

Confusion refers to making the relationship between ciphertext and key as complex as possible. *Diffusion* refers to replacing and shuffling the data, to make it impossible to analyze data statistically. This is usually done by performing substitutions and permutations in a simple pattern multiple times. This can easily be done by using an SP-network (S-box / P-box network) [Stinson, 2006, pp. 74–79]. The very first, as well as last step, of SP-Networks is usually an XOR between the subkey and the data. This is called *whitening*, and is according to Stinson [2006, p. 75] regarded as a very effective way to prevent encryption/decryption without a known key. The goal of this is to make it hard to find the key, even though one has access to multiple plaintext/ciphertext pairs produced with the same key [Shannon, 1949].

2.6.1 S-boxes

The S-box is one of the basic components that is used when creating ciphers. An S-box takes a number of input bits and creates a number of output bits in a non-linear fashion [Stinson, 2006, pp. 74–75]. They can effectively be implemented as lookup tables. Each input has to correspond to a unique output, to make sure that the input can be recreated in the descrambler.

.

2.6.2 P-Boxes

The second basic component used in cryptography is the P-box. A P-box shuffles/rearranges the order of given bits. This can be viewed in the SP-network in figure 2.4, where the P-box is represented by the dotted rectangle in the middle.

TODO:
You need sources
for this!

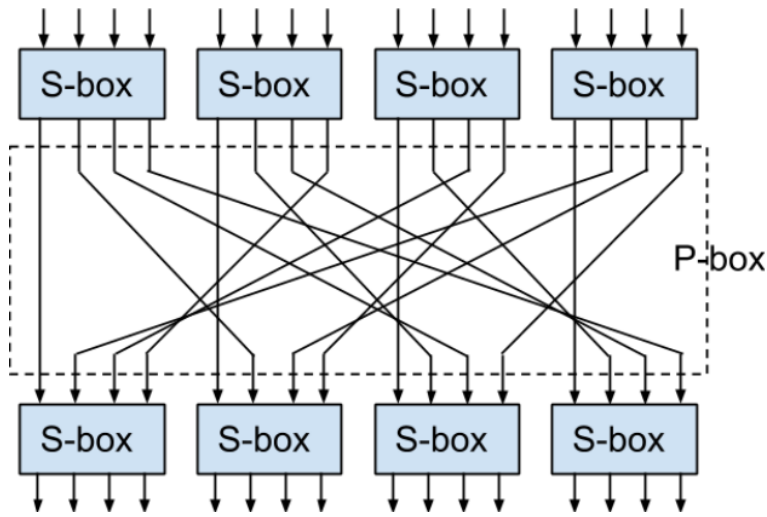


Figure 2.4: SP-Network

2.7 Secrecy

Although encryption is important, as well as the strength of the encryption, keeping the algorithm secret is never a good idea. A simple mistake when designing an algorithm might turn an encryption that would have been strong, incredibly weak. It is therefore a bad idea to use small scale algorithms (designed for the use of just a few persons for instance). If you instead use an open algorithm, faults will most likely be discovered and fixed by experienced cryptographers [Schneier and Fergusson, 2003, pp. 23]. Keeping the key, which is used to encrypt the data, secret is what is important.

3

CSA

The CSA is currently the most commonly used encryption algorithm in DVB for encryption of video-streams. There are two versions of the DVB-CSA, CSA1 and CSA2, where the key-length is the only difference between them [DVB Scene, 2013, p. 23].

The CSA uses a combination of a block cipher, taking an input of a 64-bit block, and a stream cipher. Both of the ciphers use the same key, so that the entire system uses the same key [Li and Gu, 2007, pp. 271–272]. This means that the complete algorithm would break if the key would be recovered. Using the same key does on the other hand allow us to easily change the key at regular intervals.

CSA has been the official scrambling method for DVB since may 1994. CSA was to be easily implemented in hardware and hard to implement in software to make reverse-engineering difficult [DVB Scene, 2013].

3.1 Why do we need a new standard?

The DVB-CSA standard offers short-term protection (it assumes content is viewed in real time and not stored). Due to the development of how content is viewed during recent years, we now primarily need to be able to distribute content across homes. This means that the focus needs to be moved from securing delivery to securing content. [Farncombe Consulting Group, 2009]

Another thing to bear in mind is the fact that more CPU-based units, such as smart-phones, tablets and computers are used to access contents now more than ever. In order to allow for descrambling on CPU-based units, a software-friendly scrambling algorithm might be needed.



Figure 3.1: Image from Tews et al. [2012, pp. 49]

3.2 Layout of the CSA

The CSA consists a block cipher and a stream cipher connected in sequence [Li and Gu, 2007, p. 271]. The block cipher reads 64-bit blocks of data, which is then run in Cipher Block Chaining-mode. The block cipher processes these blocks of data in 56 rounds. The output of this is sent to the stream cipher where additional encoding is performed. The first block of data sent from the block cipher to the stream cipher is used as an IV for the stream cipher, and is not encoded in this phase. [Weimann and Wirt, 2006]

3.3 Security

One of the problems associated with CW distribution is the fact that CW sharing has become rather common [Farncombe Consulting Group, 2009]. This is possibly due to the fact that the CW is sent in the clear between the smart card and the STB, meaning that a user might grab the clear CW during transmission and redistribute it over the internet. This has become a financial problem for content distributors, since people stop paying for the content which they are watching.

One way of dealing with CW sharing is to decode the encrypted CW on the CI system, and then encrypt it once again on the CI, before sending it to the STB. The latter key is setup between the CI system and the STB through a one time synchronization. This means that users are not able to grab the clear CW and redistribute it. [Schrijen, 2011, pp. 12–13]

Another security issue that you need to think of when designing the hardware, to prevent content theft, is to make sure that no contacts are ever accessible from the top layer of the circuit board. This is due to the fact that people would be

able to connect hardware to the board and download the material that way, if they were.

We also need to be aware of people trying to break the algorithm through forced ways as well as CW sharing and hardware methods of stealing content.

There are a few standard ways to try when you want to break a cipher. Those are the brute force approach, known-plaintext attacks, chosen plaintext attacks and birthday attacks [Schneier and Fergusson, 2003, pp. 31-34]. You choose what method to use depending on what the ciphers look like. I will not discuss all of them, but I will talk about the most relevant ones here.

3.3.1 Brute force

The CSA uses a key consisting of 64-bits, which gives us 18.5 Quintillion possible keys (Quintillion is 10^{18}). But byte 3 and 7 are often used as parity bytes in CA systems which leads to only 48 bits being used in the key [Tews et al., 2012]. This can be seen in figure 3.1. 48 bits on other hand leads to 2^{48} combinations, which corresponds to 281 trillion possible keys (Trillion is 10^{12}). Testing a million keys per second is about what is possible through on a modern x86 processor using software methods, which means it would take roughly 3258 days to force brake the keys. That is roughly 8.8 years.

Moreover, systems need to change the key at least every 120 seconds [Simpson et al., 2009] and most systems issues new keys every 10-120 second [Wirt, 2004].

It is possible to use dedicated hardware and FPGA implementations to speed this up, using hardware accelerations and other methods. But even if we would be able to scan through 2.8 trillion keys per second, precisely allowing us to be certain to find the key in two minutes, we could just change the key more often. As such, the brute force method of obtaining the key is not a feasible option.

Source:
Except from Patri
Lanitto

Todo:
How did I get this
number?

4

CISSA and CSA3

There are currently two scrambling algorithms being assessed as replacements to the currently used DVB-CSA. This is done to assure content security for yet another ten years.

CISSA is meant to be a hardware-friendly as well as software-friendly algorithm designed to allow descrambling to be made on CPU-based units such as computers, smart phones and tablets [ETSI TS , 2013, p. 9].

CSA3 is a hardware-friendly, software-unfriendly scrambling algorithm chosen by the ETSI to replace the currently used CSA [ETSI TS , 2013, pp. 6–7]. Software-unfriendly means that descrambling is designed so that it is highly impractical to perform in software, but easily done in hardware.

Both of the algorithms are to be implemented in hardware for scrambling of data. The difference is that CSA3 is to make it hard to descramble the material using software. Since both of the algorithms are confidential, it is sadly impossible to find out what makes the CSA3 algorithm software-unfriendly, while the CISSA algorithm is software-friendly.

Source:
Om jag får be snällt

4.1 CISSA

CISSA stands for *Common IPTV Software-oriented Scrambling Algorithm* and is designed to be software-friendly. Opposite to the CSA3, CISSA is made to be easily descrambled in software, so that CPU-based systems such as computers and smart-phones can also implement it. Although it is software-friendly, it is supposed to be able to be implemented efficiently on hardware as well as in software [ETSI TS , 2013, p. 9].

CISSA is to use the AES-128 block cipher in CBC-mode with a 16 byte IV with the value 0x445642544d4350544145534349535341. [ETSI TS , 2013, p. 11]

4.1.1 Software friendly

An FPGA implementation of the CISSA algorithm seems likely to be implementable, due to the fact that the scrambling of the content is supposed to be made in hardware, regardless as to whether the descrambling is supposed to be made either in hardware or software.

While having a scrambling algorithm designed to enable viewing on CPU-based units opens up the market for more users, it might increase the risk for algorithm theft. Since reverse-engineering is possible for software implementations, one might find the algorithm for descrambling, as well as scrambling through inversion of the algorithm. Knowing the algorithm enables cryptanalysts to search for weaknesses in the algorithm, with the purpose of breaking it.

"A cryptosystem should be secure even if everything about the system, except the key, is public knowledge." according to Kerckhoffs's Principle. This means that the only result of having a descrambling method suited for hardware as well as software implementation should possibly only result in some free implementations showing up. But it being implemented in software should therefore not lead to any problem.

4.2 CSA3

The CSA3 scrambling algorithm is based on a combination of an AES (*Advanced Encryption Standard*) block cipher using a 128-bit key, which is simply called the AES-128, and a confidential block cipher called the XRC [ETSI TS , 2013, p. 8]. XRC stands for eXtended emulation Resistant Cipher and is a confidential cipher used in DVB [ETSI TS , 2013, p. 8].

4.2.1 Hardware friendly

The CSA3 is designed to be hardware-friendly, meaning that descrambling through software methods is supposed to be next to impossible. Using a software-hostile descrambling algorithm means that reverse-engineering and algorithm theft becomes hard, if even possible. Even though it would decrease the probability of content theft, it closes the door to expansion onto the CPU-based units market, which is becoming larger and larger.

4.3 Conclusion

From what I've seen, both CISSA and CSA3 implement the AES-128 for scrambling, combined with a secret cipher. The secret cipher for CSA3 is the XRC cipher, and the secret CISSA cipher is yet to be known. It is not even sure that

Source:
No, no, not
Wikipedia

CISSA is to use a secret Cipher, but might instead use itself of merely the CBC-mode of operation for the AES128 cipher, or something.

CISSA sounds like a great idea in my opinion, allowing CPU-based units to de-scramble data streams without a dedicated HW-Chip. While that is good and all, CSA3 is a finished standard, and will probably be more easily implemented on an FPGA, while CISSA does not yet seem to be ready for the market. Starting out with an AES-128 cipher would provide for a basis to continue development of the scrambling, either towards the CISSA or the CSA3 solution, on a later stage.

5

Advanced Encryption standard

The AES is based on an SP-network and is fast both in hardware as well as software. Rijndael, which is used in AES, has key-sizes of at least 128 bits, block lengths of 128 bits, 8 to 8 bit S-boxes and a minimum of 10 rounds of repetition [Stinson, 2006, p. 79]. It is a symmetric-key algorithm with a fixed block size of 128 bits, where the key-size can vary between 128, 192 or 256 bits. The number of cycles needed to convert the plaintext into ciphertext depends on the size of the key. The 128-bit key requires 10 cycles of repetitions (rounds). The 192-bit key requires 12 rounds and the 256-bit key requires 14 rounds [Stinson, 2006, p. 103].

5.1 Method

The AES consists of a number of steps that are repeated for each block to be encoded. The steps to be performed are, according to Stinson [2006]:

Set-up steps

1. KeyExpansion - Produce round keys.
2. InitialRound - Combine each byte of the state with a byte of round key.

Steps performed in rounds

1. SubBytes - Each byte is *substituted* using the Rijndael's S-box.
2. ShiftRows - The rows of the state matrix are *permuted*.
3. MixColumns - The columns of the matrix are multiplied with a matrix.
4. AddRoundKey - The state matrix is once again combined with round-keys.

In the final round we do everything except the MixColumns step

1. SubBytes
2. ShiftRows
3. AddRoundKey

The ciphertext is then defined as the state-matrix [Stinson, 2006, p. 103]. As mentioned in section 2.6 (Confusion and Diffusion), both confusion and diffusion are necessary. They can be seen in the SubBytes and ShiftRows steps above. These steps also perform whitening, which strengthens the cipher. Whitening is, as mentioned in 2.6, performed through an XOR between the subkey and the data.

The KeySchedule is explained in section 5.2.

5.1.1 InitialRound

This is simply an initial AddRoundKey.

5.1.2 SubBytes

In the SubBytes step, each byte is sent to a Rijndael S-box (which is basically a lookup table) where they are substituted in a non-linear fashion. This gives us a substituted state matrix.

5.1.3 ShiftRows

The next step is called the ShiftRows step, which left-shifts the rows $n-1$ steps where n is the index of the row. This means that the first row is left as it is, the second row is shifted one step, the third row is shifted two steps, and the fourth row is shifted three steps.

5.1.4 MixColumns

In the MixColumns step, the four bytes of each row are combined through a matrix multiplication. The MixColumns function takes four bytes as input and multiplies them with a fixed matrix (figure A.3 in appendix A). While this might seem simple, it really is not. The multiplication makes sure that each input byte affects all output bytes. [Internet, 2014]

The matrix is multiplied from the left to the vector ($4 \times 4 * 4 \times 1 = 4 \times 1$) which is a column from the state-matrix. Multiplication with 1 means that the value is left untouched. Multiplication by 2 means left shift, then XOR with 0x1B if the value is larger than 0xFF. Multiplication with 3 means left-shift (including XOR with 0x1B if the value exceeds 0xFF), then XOR with the initial value. Addition is replaced with XOR, due to the calculations taking place in $GF(2^8)$.

5.1.5 AddRoundKey

Each of the 16 bytes of the state is combined with a byte of the round key using bitwise xor. They are then combined to a state matrix (figure A.2 in appendix A) containing 4x4 bytes.

5.2 KeyExpansion

To generate round keys from the cipher key, we use the Rijndael's key schedule. This is done since AES requires a separate 128-bit round key for each round, plus one extra key for the initialization which means that the AES-128 requires 176 bytes, since AES-128 consists of 10 rounds.

The schedule consists of a loop, and a key-schedule core. The schedule core is the part that branches out from the check if c modulo 16 is zero. The entire KeyExpansion can be viewed in Figure 5.1. To change the key schedule to fit a key size of 192 bits, you simply change the value c is compared to in the first branch in the flowchart from 176 to 206.

5.2.1 Key-schedule core

The key-schedule core takes an input of 4 bytes (32 bits) which it then rotates 1 byte (8 bits) to the left. Let us say that our key is *AB CD EF 01*. This would give us the key *CD EF 01 AB* after the rotation. This operation is also called the RotWord-operation [Stinson, 2006, p. 107]. The next step is to apply Rijndael's S-box to each of these bytes, giving us 4 new bytes. The bytes *AB CD EF 01* would give us *62 BD DF 7C* when input in the Rijndael S-box (Figure A.1).

The left-most byte is then XORed with a value from the Rcon function depending on what round you are currently at. You can read more about the Rcon function in section 5.2.3.

5.2.2 Rijndael's S-Box

Rijndael's S-box takes an input byte which it transforms according to a matrix (figure A.1 in appendix A). Where the most significant nibble is placed on the Y axis, and the least significant nibble is set on the X axis. Let us say we have an input of 0x31, then 0xC7 would be output from the Rijndael's S-box.

5.2.3 Rcon

The value input into the Rcon function depends on what round you are currently at. Which means that you would choose $Rcon(1)$ for the first round, $Rcon(2)$ for the second round, and so on. The values in the Rcon array are calculated mathematically, but might as well be accessed through a simple vector or such.

If the input value is 0 or 1, we just return that value, otherwise the following steps are performed [Wikipedia, 2014c]. This can also be replaced by an S-box

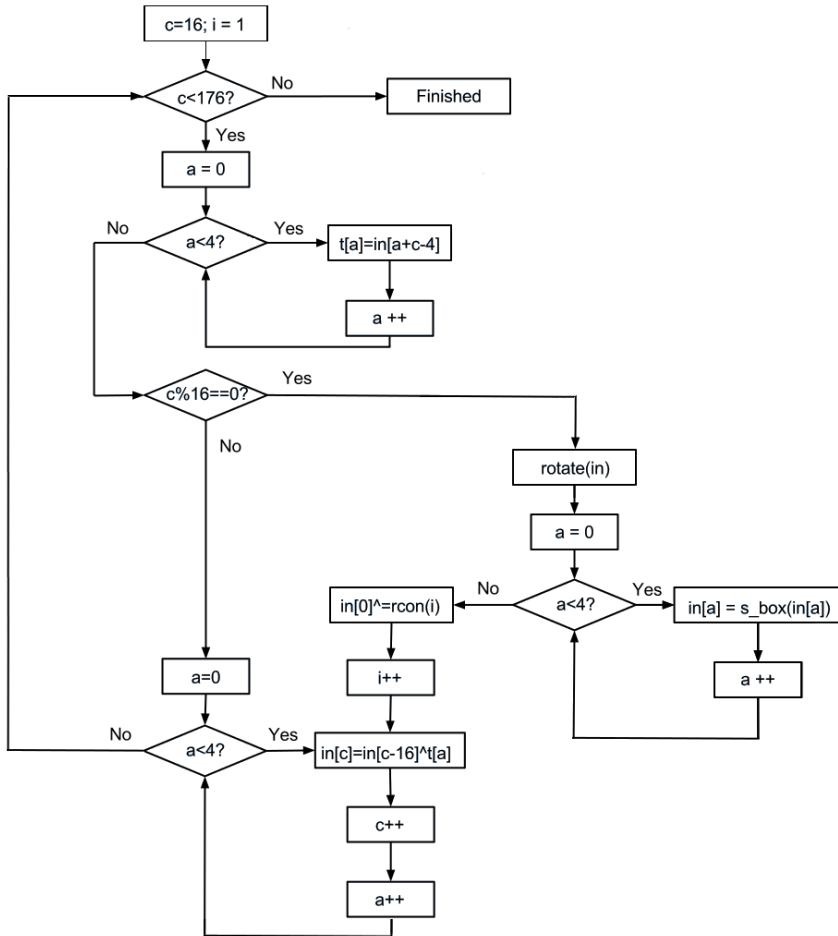


Figure 5.1: Flowchart of the key schedule

where you input your byte, and get another back, since the input byte is just used as a counter that decides how many times you perform steps 2 through 6 .

1. Set a variable c to $0x01$.
2. If the input-value does not equal 1, set variable b to $c \& 0x80$. Otherwise, go to 7.
3. Left shift c one step.
4. If b is equal to $0x80$ proceed to 5, otherwise go to 6.
5. Store the result of a bitwise XOR between c and $0x1B$ in c .
6. The input value is decreased by one, and we go back to 2.
7. We set the output to c .

TODO:
You need more reliable sources than WIKIPEDIA!

6

Result

Här är väl tanken att jag ska skriva lite om hur jag implementerat och bearbetat allt som har med CSA3 implementationen att göra.

6.1 Hardware

Såhär ser hårdvaran ut i en sjukt snygg bild jag ritat:

<i>Inputs</i>				<i>Outputs</i>		
1	--		<i>HW</i>		--	1
2	--				--	2
3	--				--	3
4	--				--	4
5	--				--	5

6.2 Flow

Berätta om flödet på implementationen.

6.3 Special solutions

Berätta om hur det fungerar.

6.4 How much of the CSA3 standard has been implemented?

Berätta om vilka delar du realiserat.

Appendix

A

Matrixes

A.1 Calculations on the CBC-mode

The ciphertext is obtained through the following equation where C_0 is the IV, and the XOR-operation is noted with \oplus .

C_i is the ciphertext

P_i is the plaintext

E_k is the encryption algorithm

D_k is the decryption algorithm

$$C_i = E_k(P_i \oplus C_{i-1}) \quad (\text{A.4})$$

The inverse of the encryption algorithm E_k is the decryption algorithm D_k .

The inverse of the XOR-operation the XOR-operation.

This gives us:

$$D_k(C_i) = P_i \oplus C_{i-1} \quad (\text{A.5})$$

which gives us

$$P_i = D_k(C_i) \oplus C_{i-1} \quad (\text{A.6})$$

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

(A.1)

Figure A.1: Rijndael S-box

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} \end{bmatrix} \quad (\text{A.2})$$

Figure A.2: State-Matrix

$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \begin{bmatrix} a_{1,i} \\ a_{2,i} \\ a_{3,i} \\ a_{4,i} \end{bmatrix}, i = \{1, 2, 3, 4\} \quad (\text{A.3})$$

Figure A.3: Rijndael MixColumns equation

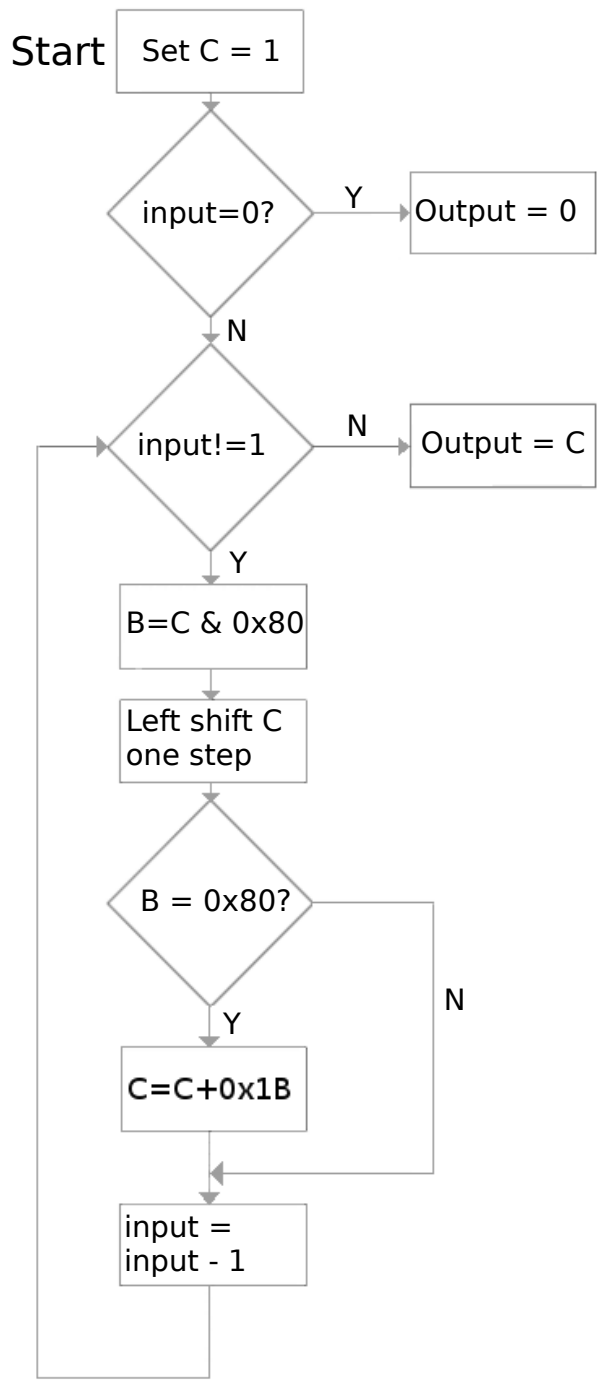


Figure A.4: Flowchart of the Rcon function

$Rcon[256] = \{$

8D	01	02	04	08	10	20	40	80	1B	36	6C	D8	AB	4D	9A
2F	5E	BC	63	C6	97	35	6A	D4	B3	7D	FA	EF	C5	91	39
72	E4	D3	BD	61	C2	9F	25	4A	94	33	66	CC	83	1D	3A
74	E8	CB	8D	01	02	04	08	10	20	40	80	1B	36	6C	D8
AB	4D	9A	2F	5E	BC	63	C6	97	35	6A	D4	B3	7D	FA	EF
C5	91	39	72	E4	D3	BD	61	C2	9F	25	4A	94	33	66	CC
83	1D	3A	74	E8	CB	8D	01	02	04	08	10	20	40	80	1B
36	6C	D8	AB	4D	9A	2F	5E	BC	63	C6	97	35	6A	D4	B3
7D	FA	EF	C5	91	39	72	E4	D3	BD	61	C2	9F	25	4A	94
33	66	CC	83	1D	3A	74	E8	CB	8D	01	02	04	08	10	20
40	80	1B	36	6C	D8	AB	4D	9A	2F	5E	BC	63	C6	97	35
6A	D4	B3	7D	FA	EF	C5	91	39	72	E4	D3	BD	61	C2	9F
25	4A	94	33	66	CC	83	1D	3A	74	E8	CB	8D	01	02	04
08	10	20	40	80	1B	36	6C	D8	AB	4D	9A	2F	5E	BC	63
C6	97	35	6A	D4	B3	7D	FA	EF	C5	91	39	72	E4	D3	BD
61	C2	9F	25	4A	94	33	66	CC	83	1D	3A	74	E8	CB	8D

 $\}$

Figure A.5: The Rcon function represented as a vector

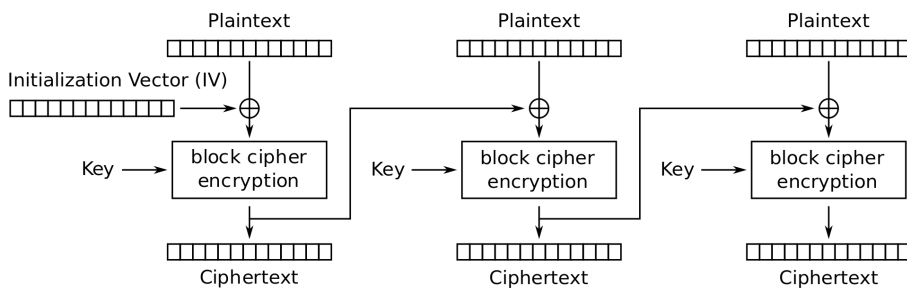


Figure A.6: Cipher block chaining mode, [Wikipedia, 2014a]

Bibliography

- Conax, 2014. URL <http://www.conax.com/about-us/>. Accessed: 13 Feb 2014. Cited on page 2.
- DVB Scene. Delivering the digital standard not so sure about the title. *DVB Scene*, September 2013. URL <http://www.dvb.org/resources/public/scene/DVB-SCENE42.pdf>. Accessed: 10 Feb 2014. Cited on page 13.
- ETSI. Digital video broadcasting (dvb); support for use of scrambling and conditional access (ca) withing digital video broadcasting systems. *ETR 289*, October 1996. URL http://www.etsi.org/deliver/etsi_etr/200_299/289/01_60/etr_289e01p.pdf. Accessed: 21 Feb 2014. Cited on page 7.
- ETSI TS. Digital video broadcasting (dvb); head-end implementation of dvb simulcrypt. *ETSI TS 103 197*, 10 2008. Accessed: 13 Feb 2014. Cited on pages 2 and 3.
- ETSI TS . Digital video broadcasting (dvb). *ETSI TS 103 127*, 05 2013. URL http://www.etsi.org/deliver/etsi_ts/103100_103199/103127/01.01.01_60/ts_103127v010101p.pdf. Accessed: 10 Feb 2014. Cited on pages 6, 7, 17, and 18.
- European Standard. Common interface specification for conditional access and other digital video broadcasting decoder applications. *EN 50221*, February 1997. URL <http://www.dvb.org/resources/public/standards/En50221.V1.pdf>. Accessed: 4 March 2014. Cited on page 3.
- Farncombe Consulting Group. Towards a replacement for the dvb common scrambling algorithm. *Farncombe White Paper*, October 2009. URL <http://farncombe.eu/whitepapers/FTLCAWhitePaperTwo.pdf>. Accessed: 28 jan 2014. Cited on pages 13 and 14.
- Simon Fischer. Analysis of lightweight stream ciphers. *École Polytechnique Fédérale De Lausanne*, 2008. URL http://biblion.epfl.ch/EPFL/theses/2008/4040/EPFL_TH4040.pdf. Accessed: 3 Feb 2014. Cited on page 10.

- Mr Internet. Mixcolumns step for aes. *Empty*, January 2014. URL http://www.angelfire.com/biz7/atleast/mix_columns.pdf. Accessed: 28 Jan 2014. Cited on page 22.
- Wei Li and Dawu Gu. Security analysis of dvb common scrambling algorithm, 2007. URL <http://ieeexplore.ieee.org.lt.ltag.bibl.liu.se/stamp/stamp.jsp?tp=&arnumber=4402690>. Accessed: 12 Feb 2014. Cited on pages 13 and 14.
- CI Plus LLP. Ci plus overview. *Common Interface Plus*, November 2011. URL http://www.ci\discretionary{-}{-}{-}plus.com/data/ci\discretionary{-}{-}{-}plus_overview_v2011\discretionary{-}{-}{-}11\discretionary{-}{-}{-}11.pdf. Accessed: 4 March 2014. Cited on page 3.
- Bruce Schneier and Niels Ferguson. *Practical Cryptography*. Wiley Publishing, Inc., first edition, 2003. Cited on pages 5, 8, 9, 11, and 15.
- G.J. Schrijen. Use case: Control word protection, May 2011. URL http://www.hisinitiative.org/_lib/img/Intrinsic\discretionary{-}{-}{-}ID_CWProtection_May_25.pdf. Accessed: 18 Feb, 2014. Cited on page 14.
- Claude Elwood Shannon. *Communication Theory of Secrecy Systems**. Bell System Technical Journal, 1949. URL netlab.cs.ucla.edu/wiki/files/shannon1949.pdf. Accessed: 28 Jan 2014. Cited on page 10.
- Gustavus J. Simmons. *Contemporary Cryptology*. IEEE Press, 1992. Cited on pages 8 and 10.
- Leonie R. Simpson, Matthew Hendricksen, and Wun-She Yap. Improved cryptanalysis of the common scrambling algorithm stream cipher, 2009. URL <http://eprints.qut.edu.au/27578/1/c27578.pdf>. Accessed: 12 Feb 2014. VET INTE OM DEN ÄR VÄRD ATT HA MED FÖR 1 SIFFRA. Cited on page 15.
- Douglas R. Stinson. *Cryptography : Theory and practice*. Chapman & Hall / CRC, third edition, 2006. Cited on pages 9, 10, 21, 22, and 23.
- Erik Tews, Julian Wälde, and Michael Weiner. Breaking dvb-csa. <http://hgpu.org/?p=6038>, pages 45 – 61, 2012. URL http://link.springer.com.lt.ltag.bibl.liu.se/chapter/10.1007%2F978-3-642-34159-5_4#page-14. Accessed: 3 Feb 2014. Cited on pages 14 and 15.
- Ralf-Philipp Weimann and Kai Wirt. Analysis of the dvb common scrambling algorithm, October 2006. URL <http://sec.cs.kent.ac.uk/cms2004/Program/CMS2004final/p5a1.pdf>. Accessed: 31 Jan 2014. Cited on page 14.
- Unknown Wikipedia. Cbc encryption, 2014a. URL <http://>

upload.wikimedia.org/wikipedia/commons/thumb/8/80/CBC_encryption.svg/2000px-CBC_encryption.svg.png. Accessed: 7 Feb 2014. Cited on page 34.

Unknown Wikipedia. Cipher-taxonomy, 2014b. URL <http://upload.wikimedia.org/wikipedia/en/thumb/8/85/Cipher-taxonomy.svg/500px-Cipher-taxonomy.svg.png>. Accessed: 5 Feb 2014. Cited on page 9.

Wikipedia Jr Wikipedia. Rijndael's key schedule. *Empty*, January 2014c. URL http://en.wikipedia.org/wiki/Rijndael_key_schedule. Accessed: 28 jan 2014. Cited on page 23.

Kai Wirt. Fault attack on the dvb common scrambling algorithm, 2004. URL <https://eprint.iacr.org/2004/289.pdf>. Accessed: 13 Feb 2014. Cited on page 15.

Upphovsrätt

Detta dokument hålls tillgängligt på Internet — eller dess framtida ersättare — under 25 år från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för icke-kommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida <http://www.ep.liu.se/>

Copyright

The publishers will keep this document online on the Internet — or its possible replacement — for a period of 25 years from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for his/her own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its www home page: <http://www.ep.liu.se/>