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

Department of Electrical Engineering
Linköpings universitet
SE-581 83 Linköping, Sweden

Linköpings tekniska högskola
Linköpings universitet
581 83 Linköping

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, 06 mars 2014

Avdelning, Institution
Division, Department

ISY Embedded systems
Department of Electrical Engineering
SE-581 83 Linköping

Datum
Date

2014-03-006

Språk
Language

☐ Svenska/Swedish

☒ Engelska/English

☐ ________________

Rapporttyp
Report category

☐ Licentiatavhandling

☒ Examensarbete

☐ C-uppsats

☐ D-uppsats

☐ Övrig rapport

☐ ________________

ISBN

—

ISRN

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

Serietitel och serienummer
Title of series, numbering

ISSN

—

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) på FPGA

Replacement of the DVB-CSA

Författare
Author

Gustaf Bengtz

Sammanfattning
Abstract

This report adresses why an implementation of CSA3 is needed as a replacement to the currently used scrambling standard CSA for cryptography for IPTV data streams.

It adresses the strengths and weaknesses of using CSA, as well as the need for encryption and scrambling of data in the DVB world.

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

Abbrevation	Meaning
AES	Advanced Encryption Standard
CAM	Conditional Access Module
CAS	Conditional Access System
CBC mode	Cipher block chaining mode
CC	Content Control
Ciphertext	Encrypted plaintext
CISSA	Common IPTV Software-oriented Scrambling Algorithm
CPU	Central Processing Unig
CSA	Common Scrambling Algorithm
CTR mode	Counter mode
CW	Control Word, which is a key
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	Content, data
PS	Program Stream
S-Box	Substitution-Box
STB	Set-top Box
TS	Transport Stream. Contains data
XRC	eXtended emulation Resistant Cipher

Contents

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 following parts will be treated in this thesis:

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

1.1 Set-up

*Transport Stream*s (TS) which contains data received from distributors are scrambled using a key which is called a *control word* (CW). Although the CW is required to be changed every 120th second, it is common to change it as often as every 10th seconds. Finding out just one CW has very little effect, since it will only be usable for a few seconds before it is changed. The high frequency in which the CW is changed thereby provides the system with security. The control word is generated randomly to make sure that consecutive control words are not related to each other.

The control word is sent to a *Conditional Access System* (CAS) where it is encrypted as an *Entitlement Control Message* (ECM). The CAS also generates an *Entitlement Management Message* (EMM) which tells the smart-card what content the user is allowed access to. The ECM and EMM are then sent back to the scrambler where it is attached to the scrambled TS using a multiplexer. This package is sent to a receiver, where the ECM, EMM and TS are separated. The ECM and TS are sent through the *Common Interface* (CI) to the *Conditional Access Module* (CAM), where the ECM is decrypted using the smart card. The resulting CW is then used to descramble the TS. The TS is encrypted once more if the CI is a CIPlus, otherwise it is sent in the clear back to the receiver where the data is processed before it is dispatched to the user. The CI and CIPlus as well as the extra encryption are all discussed in section 1.4.

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) consists among other of an EMM-generator (EMMG) and an ECM-generator (ECMG). 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 subscribing to as well as when the subscription ends. A TV will not broadcast any channels without receiving an EMM telling it to do so.

Source: You need to find more sources than just Patrik

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.

Remove?: Or keep?

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. Viaccess supports CI+.

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]. Conax supports CI+.

Strong

Check: Relevant? Plus source..

Strong is the CA-module used by Canal Digital. This is currently the third most used content provider in Sweden.. According to Wikipedia.. Strong supports CI+.

1.3 DVB-SimulCrypt

DVB-SimulCrypt is widespread in Europe, and works as an interface between the head-end and the CA system [ETSI TS, 2008]. SimulCrypt encourages the use of several CAS at once [ETSI TS, 2008, p. 17].

This is done by sending the same CW to many CA systems 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].

1.4.1 CIPlus

The CIPlus realizes the possibility of yet another means of protecting content which is called *Content Control*. CC is a means of encrypting the content inside of the CAM connected to the CIPlus Module. The key used for the CC encryption is paired with the Digital TV Receiver, where the TS is decrypted before being made available to users. The general idea can be viewed in Figure 1.1.

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.

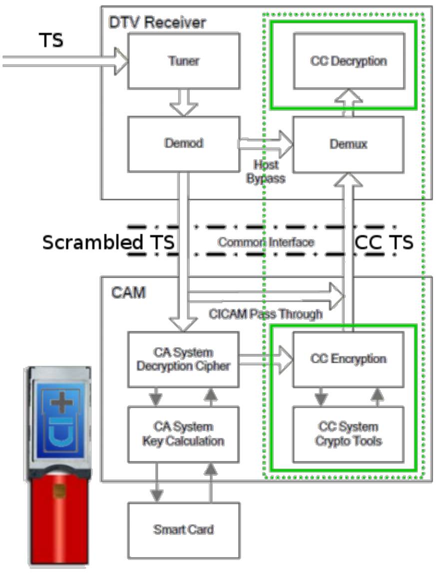

Figure 1.1: *CIPlus interface. Image remade from [LLP, 2011, p. 10]*

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 an ordinary 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. On the other hand, you might never notice that the security had been breached, if the the thief were to pick the lock instead. [Schneier and Fergusson, 2003]

One of the more noteworthy cryptography rule is that you always are to assume that someone is out to get you. Because of this, Schneier and Fergusson [2003, pp. 12–14] says that we always need to look for possible ways to break systems, to ensure that the security can not be breached, and thereby provides security.

2.1 Why do we need cryptography?

Cryptography is the science of rendering plaintexts into ciphertexts to protect contents from unauthorized viewing. It is used in electronic communication for protection of e-mail messages and credit card information among other things. If we send data without encrypting it, someone who is eavesdropping to the transmission channel will most likely access the data.

For most people this is not 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. You might also not want people to be able to read your account information and card

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

Source: Typ Schneier har jag för mig? Låna om boken?

numbers when buying things online either.

Another reason for scrambling is to reduce the number of adjacent data-bits with the same value, like strings of zeroes or ones. It could also serve to balance the number of zeroes and ones in strings. This is done as to try to obtain DC balance. DC balance is desired since it avoids voltage imbalance during communication between connected systems.

Rewrite: And find a good source for this

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. Among the three ways of packing data, only two are commonly used. PS packets are used for storing data, while TS and PES are used for transmitting data. The interresting types, 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 lenghts. 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 TS packet consists of 4 different kinds of building blocks where only the header is guaranteed to be present. Those blocks are:

- Header
- Adaptation field
- Encrypted payload

- Clear payload

The byte-sizes of the building blocks are:

- header_size = 4
- adaptation_field_size = the size of the adaptation field
- payload_size = 188 - (header_size + adaptation_field_size)
- encrypted_payload_size = payload_size - [payload_size mod block_size]
- clear_payload_size = [payload_size mod block_size] (or simply payload_size - encrypted_payload_size)

Header

The header consists of information regarding the packet, and has a sync_byte (with a hex-value of 0x47, or bit-value of 01000111) to display the beginning of the packet. The header also contains information as to whether there is an adaptation field and payload in the packet, what PID the packet has, if it is to be prioritized, whether the data is scrambled - and in that case if it was scrambled with an odd or even key, among others [ETSI TS, 2009, pp. 25–26] . The header is never to be encrypted and is always found at the beginning of a packet. [ETSI TS, 2013, pp. 10–11]

Adaptation field

The adaptation field is a sort of padding that you input when the end of the data does not align with the end of the TS packet. This is done to make sure that the TS packet is filled with known data. 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. [ETSI TS, 2013, pp. 10–11]

Encrypted and clear payload

When working with block ciphers, we tend to end up with clear bytes of data since block ciphers only encrypt data blocks of fixed sizes. The clear data is always located at the end of the TS packets and might be of sizes up to one byte smaller than the block size at the most. 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, and are often packed into TS packets when distributed, due to the strength of TS packets. The payload data in the TS packets, when carrying PES packets, consist of the entire PES packets, which is the header as well as the data. PES packets do not use adaptation fields, since they are of adaptable lengths, as long as the length of the packet does not exceed 64 KBytes.

Since DVB seldom uses itself of PES packets, an analyzation of the header will not be done. The derivation of PES packets from TS packets can be seen in Figure

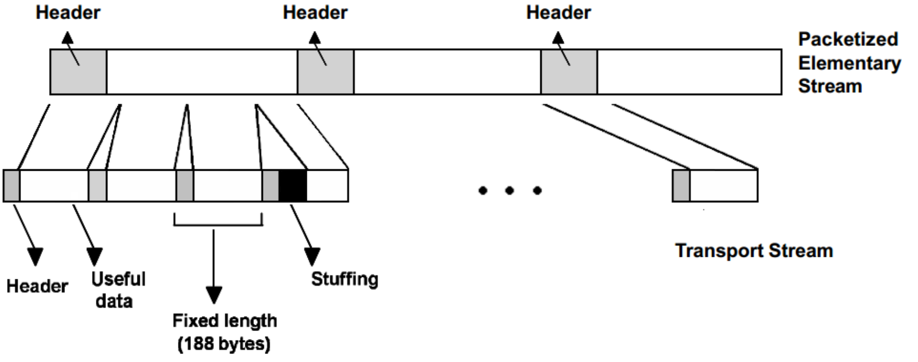

Figure 2.2: *PES packet derived from TS packets (inspiration taken from [ETSI, 1996, p. 9])*

2.2.

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. Distrubution 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].

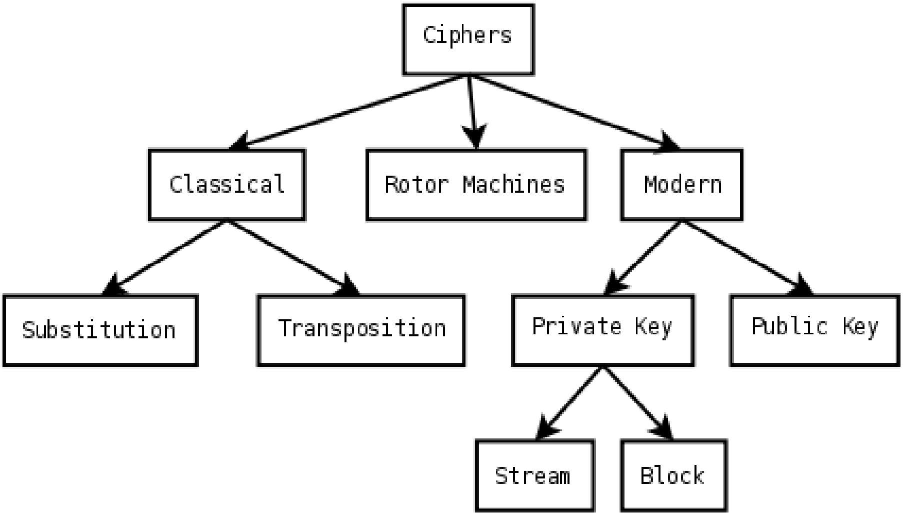

Figure 2.3: *Different kinds of ciphers [Wikipedia, 2014b]*

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 symmteric-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 D.1 in appendix D. 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.

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 CSA 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 B.3 in appendix B.

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

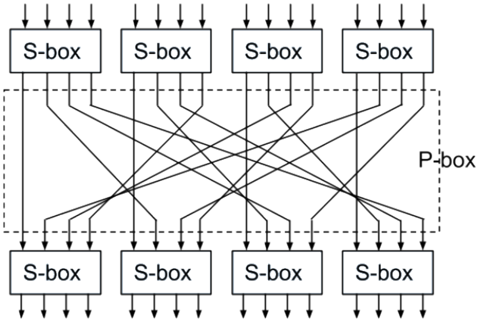

Figure 2.4: *SP-Network*

[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.

TODO: You need source for this!

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.

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.

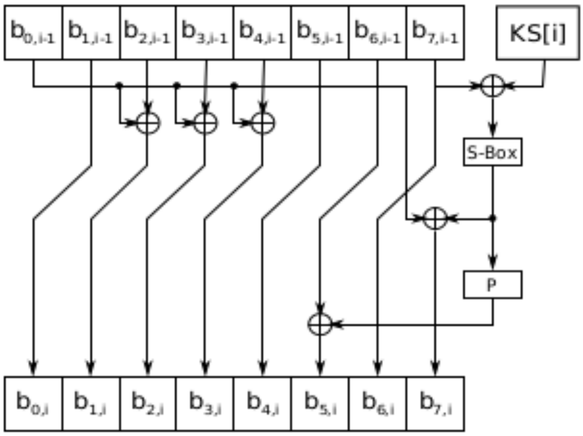

Figure 3.1: *Image from Tews et al. [2012, pp. 49]*

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 sychronization. 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.

Source: Except from Patri Lantto

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.

Todo: How did I get thi number?

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.

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 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. Each TS packet is to be process independetly of other TS packets, but each block of data in the payload depends on the previous blocks of data in the same payload. Both the header and adaptation field are to be left unscrambled. [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 cryptoanalysists to search for weaknesses in the algorithm, with the purpose of breaking it.

Source: No no, not Wikipedia

"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 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 descramble 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 chiper 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 *permutated*.
3. MixColums - The columns of the matrix are multiplicated 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 nescessary. They can be seen in the SubBytes and ShiftRows steps above. These steps also performs 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-shift 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 affect all output bytes. [Internet, 2014]

The matrix is multiplicated from the left to the vector (4x4 * 4x1 = 4x1) 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 is 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) .

DOODOO: Add a reference to Galois fields

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 B.2 in appendix B. 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 (A.4 in appendix A).

I illustrated the steps to be performed in the Rcon function using a flowchart which can be viewed in figure B.1 in appendix B.

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 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 .

TODO: You need more reliable sources than WIKIPEDIA!

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.

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:

$$
\begin{array}{rlcll}
Inputs & & \underline{\qquad\qquad} & & Outputs \\
1 & --| & & |-- & 1 \\
2 & --| & & |-- & 2 \\
3 & --| & HW & |-- & 3 \\
4 & --| & & |-- & 4 \\
5 & --| & & |-- & 5 \\
& & \underline{\qquad\qquad} & &
\end{array}
$$

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

Nibble	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*

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} \end{bmatrix} \tag{A.2}$$

Figure A.2: *State-Matrix*

$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \begin{bmatrix} a_{1,i} \\ a_{2,i} \\ a_{3,i} \\ a_{4,i} \end{bmatrix}, i = \{1, 2, 3, 4\} \tag{A.3}$$

Figure A.3: *Rijndael MixColumns equation*

$$
Rcon[256] = \{ \begin{array}{llllllllllllllll}
8D & 01 & 02 & 04 & 08 & 10 & 20 & 40 & 80 & 1B & 36 & 6C & D8 & AB & 4D & 9A \\
2F & 5E & BC & 63 & C6 & 97 & 35 & 6A & D4 & B3 & 7D & FA & EF & C5 & 91 & 39 \\
72 & E4 & D3 & BD & 61 & C2 & 9F & 25 & 4A & 94 & 33 & 66 & CC & 83 & 1D & 3A \\
74 & E8 & CB & 8D & 01 & 02 & 04 & 08 & 10 & 20 & 40 & 80 & 1B & 36 & 6C & D8 \\
AB & 4D & 9A & 2F & 5E & BC & 63 & C6 & 97 & 35 & 6A & D4 & B3 & 7D & FA & EF \\
C5 & 91 & 39 & 72 & E4 & D3 & BD & 61 & C2 & 9F & 25 & 4A & 94 & 33 & 66 & CC \\
83 & 1D & 3A & 74 & E8 & CB & 8D & 01 & 02 & 04 & 08 & 10 & 20 & 40 & 80 & 1B \\
36 & 6C & D8 & AB & 4D & 9A & 2F & 5E & BC & 63 & C6 & 97 & 35 & 6A & D4 & B3 \\
7D & FA & EF & C5 & 91 & 39 & 72 & E4 & D3 & BD & 61 & C2 & 9F & 25 & 4A & 94 \\
33 & 66 & CC & 83 & 1D & 3A & 74 & E8 & CB & 8D & 01 & 02 & 04 & 08 & 10 & 20 \\
40 & 80 & 1B & 36 & 6C & D8 & AB & 4D & 9A & 2F & 5E & BC & 63 & C6 & 97 & 35 \\
6A & D4 & B3 & 7D & FA & EF & C5 & 91 & 39 & 72 & E4 & D3 & BD & 61 & C2 & 9F \\
25 & 4A & 94 & 33 & 66 & CC & 83 & 1D & 3A & 74 & E8 & CB & 8D & 01 & 02 & 04 \\
08 & 10 & 20 & 40 & 80 & 1B & 36 & 6C & D8 & AB & 4D & 9A & 2F & 5E & BC & 63 \\
C6 & 97 & 35 & 6A & D4 & B3 & 7D & FA & EF & C5 & 91 & 39 & 72 & E4 & D3 & BD \\
61 & C2 & 9F & 25 & 4A & 94 & 33 & 66 & CC & 83 & 1D & 3A & 74 & E8 & CB & 8D\}
\end{array}
$$

Figure A.4: *The Rcon function represented as a vector*

B

Illustrations

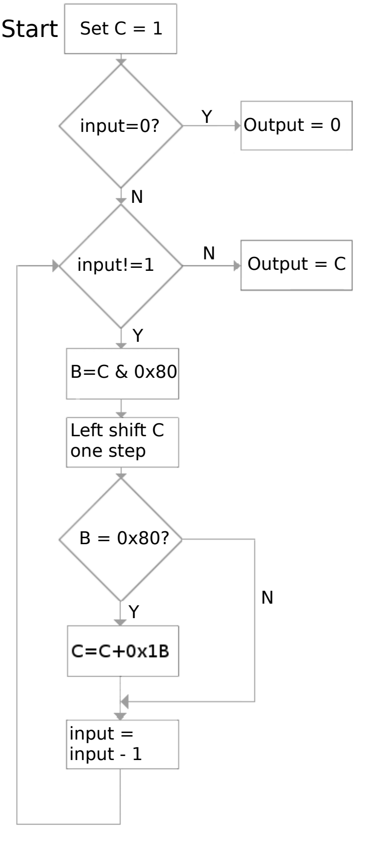

Figure B.1: *Flowchart of the Rcon function*

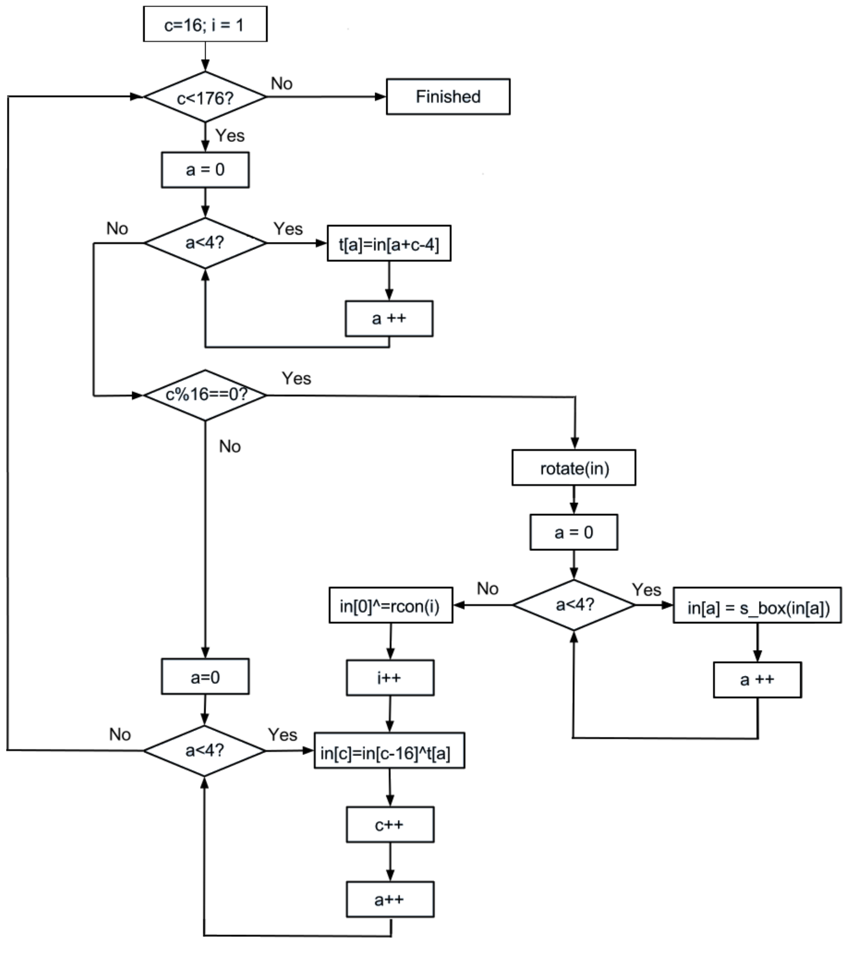

Figure B.2: *Flowchart of the key schedule*

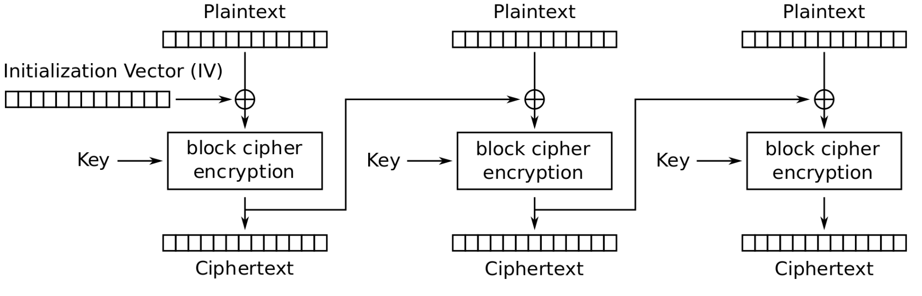

Figure B.3: *Cipher block chaining mode, [Wikipedia, 2014a]*

C
Test vectors

Test vector taken from NIST [2001, pp. 35–36].
AES-128 (Nk=4, Nr=10)

PLAINTEXT:	00112233445566778899AABBCCDDEEFF
KEY:	000102030405060708090A0B0C0D0E0F

CIPHER (ENCRYPT):

round[0].input	00112233445566778899AABBCCDDEEFF
round[0].k_sch	000102030405060708090A0B0C0D0E0F
round[1].start	00102030405060708090A0B0C0D0E0F0
round[1].s_box	63CAB7040953D051CD60E0E7BA70E18C
round[1].s_row	6353E08C0960E104CD70B751BACAD0E7
round[1].m_col	5F72641557F5BC92F7BE3B291DB9F91A
round[1].k_sch	D6AA74FDD2AF72FADAA678F1D6AB76FE
round[2].start	89D810E8855ACE682D1843D8CB128FE4
round[2].s_box	A761CA9B97BE8B45D8AD1A611FC97369
round[2].s_row	A7BE1A6997AD739BD8C9CA451F618B61
round[2].m_col	FF87968431D86A51645151FA773AD009
round[2].k_sch	B692CF0B643DBDF1BE9BC5006830B3FE
round[3].start	4915598F55E5D7A0DACA94FA1F0A63F7
round[3].s_box	3B59CB73FCD90EE05774222DC067FB68
round[3].s_row	3BD92268FC74FB735767CBE0C0590E2D
round[3].m_col	4C9C1E66F771F0762C3F868E534DF256
round[3].k_sch	B6FF744ED2C2C9BF6C590CBF0469BF41
round[4].start	FA636A2825B339C940668A3157244D17
round[4].s_box	2DFB02343F6D12DD09337EC75B36E3F0

round[4].s_row	2D6D7EF03F33E334093602DD5BFB12C7
round[4].m_col	6385B79FFC538DF997BE478E7547D691
round[4].k_sch	47F7F7BC95353E03F96C32BCFD058DFD
round[5].start	247240236966B3FA6ED2753288425B6C
round[5].s_box	36400926F9336D2D9FB59D23C42C3950
round[5].s_row	36339D50F9B539269F2C092DC4406D23
round[5].m_col	F4BCD45432E554D075F1D6C51DD03B3C
round[5].k_sch	3CAAA3E8A99F9DEB50F3AF57ADF622AA
round[6].start	C81677BC9B7AC93B25027992B0261996
round[6].s_box	E847F56514DADDE23F77B64FE7F7D490
round[6].s_row	E8DAB6901477D4653FF7F5E2E747DD4F
round[6].m_col	9816EE7400F87F556B2C049C8E5AD036
round[6].k_sch	5E390F7DF7A69296A7553DC10AA31F6B
round[7].start	C62FE109F75EEDC3CC79395D84F9CF5D
round[7].s_box	B415F8016858552E4BB6124C5F998A4C
round[7].s_row	B458124C68B68A014B99F82E5F15554C
round[7].m_col	C57E1C159A9BD286F05F4BE098C63439
round[7].k_sch	14F9701AE35FE28C440ADF4D4EA9C026
round[8].start	D1876C0F79C4300AB45594ADD66FF41F
round[8].s_box	3E175076B61C04678DFC2295F6A8BFC0
round[8].s_row	3E1C22C0B6FCBF768DA85067F6170495
round[8].m_col	BAA03DE7A1F9B56ED5512CBA5F414D23
round[8].k_sch	47438735A41C65B9E016BAF4AEBF7AD2
round[9].start	FDE3BAD205E5D0D73547964EF1FE37F1
round[9].s_box	5411F4B56BD9700E96A0902FA1BB9AA1
round[9].s_row	54D990A16BA09AB596BBF40EA111702F
round[9].m_col	E9F74EEC023020F61BF2CCF2353C21C7
round[9].k_sch	549932D1F08557681093ED9CBE2C974E
round[10].start	BD6E7C3DF2B5779E0B61216E8B10B689
round[10].s_box	7A9F102789D5F50B2BEFFD9F3DCA4EA7
round[10].s_row	7AD5FDA789EF4E272BCA100B3D9FF59F
round[10].k_sch	13111D7FE3944A17F307A78B4D2B30C5
round[10].output	69C4E0D86A7B0430D8CDB78070B4C55A

D Examples

D.1 CBC-mode calculations

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

$$C_i = E_k(P_i \oplus C_{i-1}) \tag{D.1}$$

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:

$$D_k(C_i) = P_i \oplus C_{i-1} \tag{D.2}$$

which gives us

$$P_i = D_k(C_i) \oplus C_{i-1} \tag{D.3}$$

List of Figures

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 pages 8 and 41.

ETSI TS. Digital video broadcasting (dvb); head-end implementation of dvb simulcrypt. *ETSI TS 103 197*, 10 2008. Accessed: 13 Feb 2014. Cited on page 3.

ETSI TS. Digital video broadcasting (dvb); specification for the use of video and audio coding in broadcasting applications based on the mpeg-2 transport stream. *ETSI TS 101 154*, 9 2009. URL http://www.etsi.org/deliver/etsi_ts/101100_101199/101154/01.09.01_60/ts_101154v010901p.pdf. Accessed: 6 March 2014. Cited on page 7.

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, 18, and 41.

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 pages 3, 4, and 41.

NIST. Specification for the advanced encryption standard (aes), November 2001. URL http://csrc.nist.gov/publications/fips/fips197/fips\discretionary{-}{}{}197.pdf. Accessed: 17 Feb 2014. Looks like shit but wth. Cited on page 37.

Bruce Schneier and Niels Fergusson. *Practical Cryptography*. Wiley Publishing, Inc., first edition, 2003. Cited on pages 5, 8, 10, 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 pages 10 and 11.

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 10, 11, 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, 15, and 41.

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 pages 35 and 41.

Unknown Wikipedia. Cipher-taxanomy, 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 pages 9 and 41.

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 24.

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.