

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 (AES128)
på FPGA**

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 <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						
<table border="0"> <tr> <td data-bbox="194 638 315 906">Titel Title</td> <td data-bbox="315 638 1156 906"> Genomgång av nya och alternativa krypterings- och scramblingsystem för digital-teve samt implementering av ny scrambling-algoritm (AES128) på FPGA Replacement of the DVB-CSA </td> </tr> <tr> <td data-bbox="194 815 315 906">Författare Author</td> <td data-bbox="315 815 1156 906">Gustaf Bengtz</td> </tr> </table>			Titel Title	Genomgång av nya och alternativa krypterings- och scramblingsystem för digital-teve samt implementering av ny scrambling-algoritm (AES128) på FPGA Replacement of the DVB-CSA	Författare Author	Gustaf Bengtz
Titel Title	Genomgång av nya och alternativa krypterings- och scramblingsystem för digital-teve samt implementering av ny scrambling-algoritm (AES128) på FPGA Replacement of the DVB-CSA					
Författare Author	Gustaf Bengtz					
Sammanfattning Abstract <p>This report addresses why the currently used scrambling standard CSA needs a replacement. Proposed replacements to CSA are analyzed to some extent, and an alternative replacement (AES) is analyzed.</p> <p>It has been impossible to find proper information on CISSA and CSA3 due to them being confidential, and licensing was not allowed.</p> <p>The implementation of the Advanced Encryption Standard (AES) is analyzed, and the general implementation is displayed.</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
CC	Content Control
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, 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
FF	Flip-Flop
IPTV	Internet Protocol Television
IV	Initialization vector
LFSR	Linear Feedback Shift-Register
LSB	Least Significant Bit
LUT	Look-up Table
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

Notation	v
1 Digital Video Broadcasting	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
1.4.1 CIPlus	3
1.5 Conditional Access Module	4
2 Scrambling	5
2.1 Why do we need 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	8
2.4.1 Symmetric-key encryption	8
2.4.2 Public-key encryption	8
2.4.3 Combination	9
2.5 Ciphers	9
2.5.1 Block cipher	10
2.5.2 Stream cipher	10
2.6 Confusion and Diffusion	10
2.6.1 S-boxes	11
2.6.2 P-Boxes	11
2.7 Secrecy	11
3 Common Scrambling Algorithm	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	19
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	25
6.1	Problems	25
6.2	Hardware	26
6.2.1	Hardware usage	26
6.3	Further development	27
6.3.1	VETINTE	27
6.3.2	SBox	27
6.4	Implementation	28
6.4.1	Manager entity	28
6.4.2	CBC entity	28
6.4.3	Cipher entity	28
6.4.4	Keyexpansion entity	29
6.4.5	Round entity	29
6.5	Tests	30
A	Matrixes	35
B	Illustrations	39
C	Test vectors	43
C.1	Test cases	43
C.2	Keyexpansion	48
D	Examples	51
D.1	CBC-mode calculations	51

E Layout of the circuit	53
---------------------------	----

List of Figures	59
-----------------	----

Bibliography	61
--------------	----

1

Digital Video Broadcasting

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 Streams (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 pack-

age 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.

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.

CA system	Used by	Supports CI+
Viaccess	Boxer, SVT	Yes
Conax	Com Hem	Yes
Strong	Canal Digital	Yes

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

Source:
You need to find more sources than just Patrik

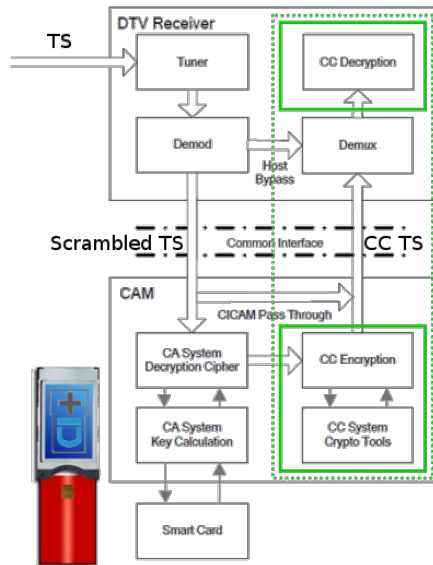


Figure 1.1: CIPlus interface. Image remade from [LLP, 2011, p. 10]

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.

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

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.

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.

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 interesting 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 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[ETSI TS, 2013].

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:

- $\text{header_size} = 4$
- $\text{adaptation_field_size} = \text{the size of the adaptation field}$
- $\text{payload_size} = 188 - (\text{header_size} + \text{adaptation_field_size})$
- $\text{encrypted_payload_size} = \text{payload_size} - [\text{payload_size} \bmod \text{block_size}]$
- $\text{clear_payload_size} = [\text{payload_size} \bmod \text{block_size}]$ (or simply $\text{payload_size} - \text{encrypted_payload_size}$)

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

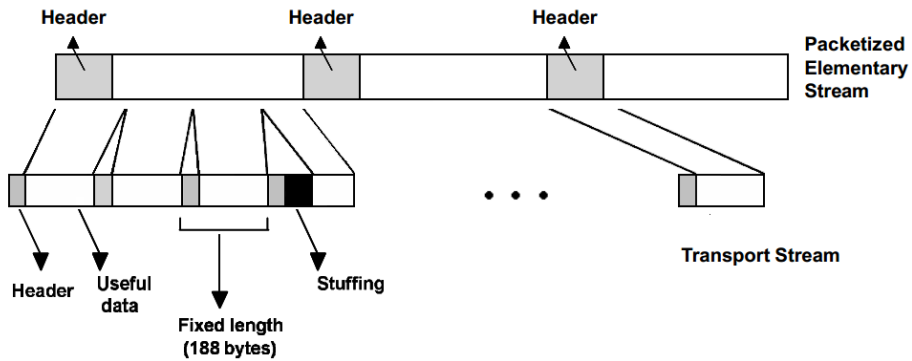


Figure 2.2: PES packet derived from TS packets

2.2 [ETSI, 1996, p. 9].

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].

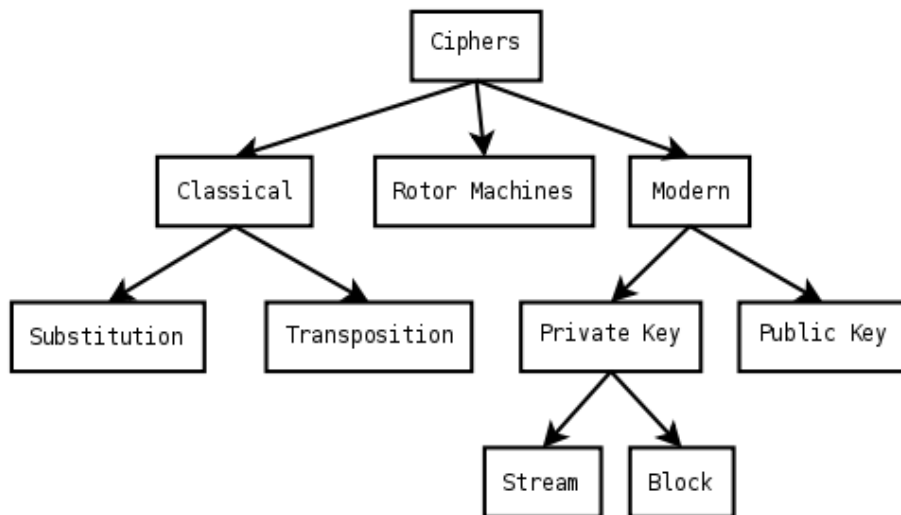


Figure 2.3: Different kinds of ciphers [Wikipedia, 2014b]

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 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 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 the different kinds of ciphers can be split into sub-groups.

Decision
Relevance?

There are mainly two kinds of ciphers that are used when designing modern cryp-

tosystems. 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 [Vaudenay et al., 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

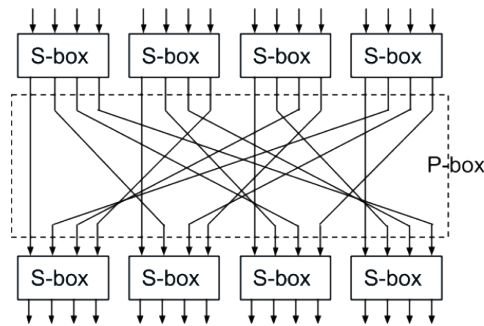


Figure 2.4: SP-Network

[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 sources
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

Common Scrambling Algorithm

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, 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.

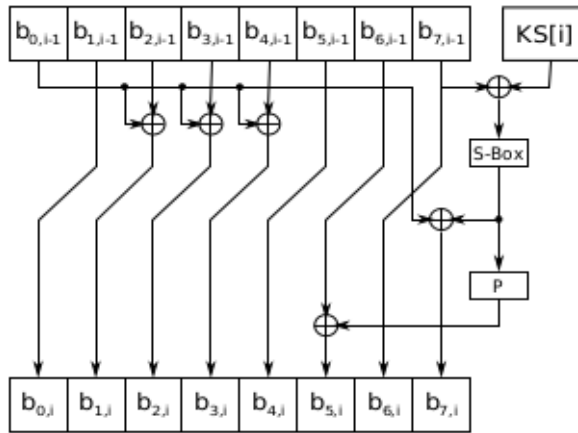


Figure 3.1: Number of bits in key used

3.2 Layout of the CSA

The CSA consists a block cipher and a stream cipher connected in sequence [Li, 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. [Weinmann 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. Each TS packet is to be processed independently of other TS packets, but each block of data in the payload depends on the previous blocks of data in the same payload, except the first block of data, which depends on the IV. 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 cryptanalysts 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

Both CISSA and CSA3 seem to implement the AES-128 for scrambling, combined with some kind of a secret cipher. The secret cipher for CSA3 is the XRC cipher, while even the name of CISSAs secret cipher has been hard to find. 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.

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 with the vector from the left, ($4 \times 4 \times 4 \times 1 = 4 \times 4 \times 4 \times 1 = 4 \times 1$) where the vector is a column from the state-matrix. Multiplication with 1 means that the value is left untouched. Multiplication by 2 means left shift, then an XOR with 0x1B if the shifted value exceeds 0xFF. Multiplication with 3 is done in the same way as a multiplication with 2, except that the result after the shift and conditional XOR are then XOR'ed with the input value of the multiplication. All of the resulting values are then XOR'ed, leaving us with the result. All additions are replaced with XOR, since the calculations take place in $GF(2^8)$ (Galois field).

5.1.5 AddRoundKey

Each of the 16 bytes of the state are then combined with a byte from the round key using a 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 (16-byte) 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 couple of loops and a key-schedule core. The schedule core is the part that branches out 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 substituted according to the Rijndael S-box (Figure A.1).

The left-most byte is then XOR:ed with a value from the Rcon function depending on what round you are currently processing. 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 LUT (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. Given the input *0x31*, we would receive an output of *0xC7* 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 from a vector, such as the one found in Figure 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 .

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

The focus of my implementation has been to minimize the amount of hardware usage, while meeting the timing constraints provided from the rest of the circuit. The clock frequency used on the FPGA has been 100 MHz. A throughput in the scale of Gbits/s is sufficient for the current design.

The implemented scrambler processes 16 bytes of data in 11 clock pulses with a clock frequency of roughly 100MHz, which would correspond roughly to a throughput of 1.16 Gbits/s. I don't know if it can deal with 100MHz just yet, but probably. The scrambler needs to process the key before scrambling input data. A keyexpansion takes roughly 45 clock pulses, and is only performed when a new key is sent, which is very seldom.

Här är väl tanken att jag ska skriva lite om hur jag implementerat och bearbetat allt relevant som har med implementationen att göra. Vad jag gjort som blivit bättre än andra lösningar. Vad jag fokuserat på (throughput kontra mängd använd hårdvara).

6.1 Problems

Encountered problems:

- Not possible to get the license for CSA3
- Small interest for CSA3
- Next to no documentation of the CISSA algorithm
- Hard finding reliable test vectors

- Merging
- Timing
- Latches

Most of these problems are, in my mind, self-explanatory. The one that I will discuss here is my problem with merging. This problem occurred due to the fact that I started the project by implementing small entities, that were to be used in higher hierarchies, instead of what signals would be needed to be sent from different entities to each other, as well as what they would actually mean.

The pro of my method of working has been that I have been able to get results quickly. The con is that a large portion of the time has been spent on going back to entities that were already functional, and reworking them by adding signals, and finding the right timing conditions to make sure that they provided necessary information for entities higher up in the hierarchy.

Since I tried to optimize this implementation to just meet the demands on speed, while trying to minimize the amount of hardware needed, I introduced timing into a circuit that could otherwise be completely combinatorial. This has, as expected, introduced quite a bunch of timing-issues. I dare say that all of them are gone now, but it is quite hard to know without testing the circuit more extensively.

When I first synthesized the circuit, towards the end of the implementation, I found that the circuit synthesized a large amount of latches. This made my circuit take up roughly 15% of the FPGA, and use 11830 Flip-Flops. There were roughly 3000 latches. When I managed to remove all of them, my entire circuit used roughly 8% of the FPGA, and used about 4500 Flip-Flops instead.

This is the entire hardware usage, including the interface towards the FPGA, which is one of the reasons why it might appear large, when compared to other implementations.

6.2 Hardware

The top entity can be viewed in Figure 6.1, and the rest of the entities can be viewed in appendix E.

6.2.1 Hardware usage

I have run three rounds of synthesis on my circuit. The first two ones were performed on the entire scrambler, including the manager (interface towards the rest of the FPGA), while the last synthesis were performed on each block of the circuit separately.

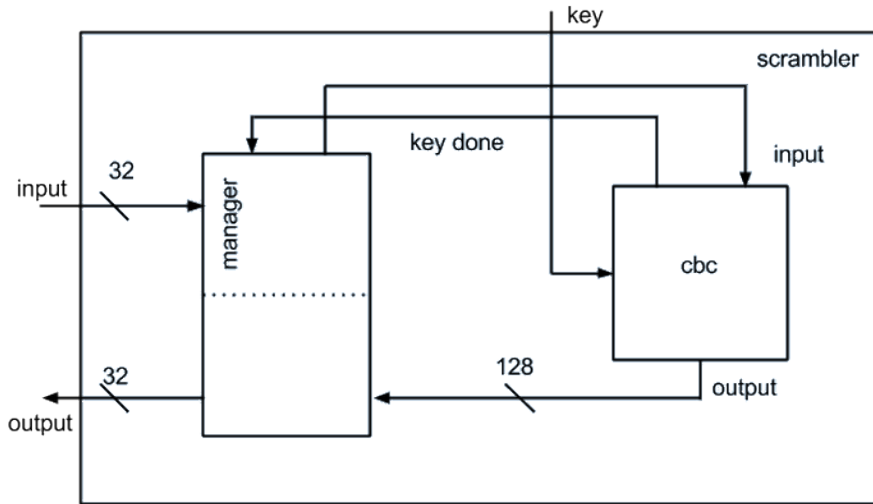


Figure 6.1: The top entity

First synthesis

The circuit after the first synthesis used up 15% of the FPGA, and had quite a large amount of unnecessary latches and FFs included. It used 11830 FFs, and roughly 3000 latches. I re-designed the circuit to remove the latches and ran the second synthesis.

Second synthesis

The second synthesis report said that 8% of the FPGA was occupied by the circuit. The keyblock3 entity, as well as keyblock1 entity seemed to use a lot of registers and multiplexers, which could possibly be replaced by RAMs or LUTs.

6.3 Further development

There are, as usual, an amount of optimization that could be performed on the circuit. They consist of optimization of code, as well as some deeper research into how to write VHDL code to turn the registers in this implementation into RAMs, ROMs or LUTs.

6.3.1 VETINTE

6.3.2 SBox

The Rijndael Sbox implemented in my design does not synthesize into a ROM, which it should be able to do, alternatively into a couple of LUT6. I have not been able to find out why my code is implemented into registers, but it is. It would

also be possible to implement this into a set of four LUTs, and only run one at the time. OR SOMETHING.

Check:
If this is true!

6.4 Implementation

My design is very hierarchical. The top layer is an aes128 block in CBC-mode. It takes an input TS-packet, selects data from it which it scrambles, and then outputs the data in the form of a TS-packet once again.

The scrambler consists of two entities. An entity which I call the cbc-entity, which deals with the scrambling of the received data. The other entity is a data-manager. The manager deals with reading data from the interface towards the rest of the FPGA as well as sending the right data-bits to the CBC-entity. It also tells the CBC-entity how to handle the data, since different things are to be done depending on if the data is the first data packet sent, or not.

6.4.1 Manager entity

The manager (Figure E.2) consists of a FIFO, an FSM and a couple of registers. The FIFO is needed since the data sent to the scrambler from the FPGA is sent in bursts. The FIFO therefore writes the data bursts into a memory, from which it later reads, processes and sends the data to the CBC-entity. The data written to the FIFO is written in packets of 32 bits, but are read 8 bits at the time. The manager looks through the data packets to see if there is an adaptation field or not, since that changes the way we handle the data. The payload is written to the first set of registers as the data is found, and then sent to the next set of registers. This is simply done to allow the manager to deal with two sets of data in parallel. When the packet is ready to be sent, a flag is set and the data is sent to the CBC-entity.

6.4.2 CBC entity

The CBC-entity (Figure E.3) consists of three small entities. An XOR, a multiplexer and a cipher-entity. The multiplexer is needed since we want to input the first plaintext into the XOR together with an IV. We want to use the output ciphertext instead of the IV for the rest of the plaintexts contained within the same TS-packet. There is only going to be one aes128 cipher in the CBC-entity, in order to save hardware. It will be run in sequence instead of in parallel, even though it might reduce the maximal speed of the circuit.

6.4.3 Cipher entity

The aes-128 cipher-entity (Figure E.4) consists of 4 components. The data2state entity, which transforms the array into a matrix of data. A keyexpansion entity, which takes an input of a key, and generates an extended key as an output. An entity, which I chose to call rounds, which deals with the encryption of the 16 byte blocks. And finally a state2data entity, which transforms the data-matrix into an array once again. The cipher entity itself keeps track of timing mainly

between the keyexpansion and the round entity, and makes sure to provide the round entity with the correct roundkey at the right time.

6.4.4 Keyexpansion entity

The keyexpansion-entity is divided into 3 keyblock entities. The first keyblock entity decides what 4 bytes of the expanded key we want to expand. The second keyblock entity is the keycore, which is only performed on every 4th set of 4 bytes. The third keyblock entity performs an xor and an incrementation of the internal counter used as an index when accessing 4 byte blocks of data.

Keycore entity

The keycore entity consists of four entities. Rotword, Sbox, Rcon and a counter. The counter is used to get the right data-byte from the Rcon entity, and the index is only used in the keycore, and is thus best suited to be placed inside the keycore entity. Rotword rotates the bytes of the input one step to the left. Sbox replaces the input bytes according to the Rijndael Sbox. The Rcon entity both collects the correct rcon value from a precalculated vector, as well as inputs it into an xor together with the input.

6.4.5 Round entity

The round-entity (Figure E.5) consists of four entities. Subbytes, shiftrows, mixcolumns and addroundkey. Addroundkey is a somewhat special XOR. Subbytes is an Rijndael Sbox, which takes an input 16-byte state, substitutes it, and outputs another 16-byte state. Shiftrows transposes the rows of the second, third and fourth row of the state. Last, but not least, is the mixcolumns entity. It consists of 16 mulblock entities. The input state of mixcolumns is split into columns, and each column is sent to a mulblock entity, which multiplies the inputs with 1, 2 or 3, then performs a bitwise XOR on them, outputting the result of the XOR. The function of the mixcolumns block is a rather complex matrix multiplication.

Addroundkey entity

Addroundkey is an entity which takes different inputs depending on what round we are currently dealing with. On the first round, addroundkey takes the input to the round entity. On the last round, it takes the output from the subbytes entity. The input to addroundkey is the output from mixcolumns the rest of the time.

The mulblock entity

The mulblock entity consists of one mul3 entity and one mul2 entity, which performs a special kind of hardware multiplication of 3, and 2, on the input. It also takes two inputs which it leaves alone. The four results are then XOR:ed with eachother, and returned to the mixcolumns entity. The result is then input into the correct index in the matrix.

Mul3 means multiplication with 3, and mul2 means multiplication with 2. A multiplication with 2 is a left-shift, followed by an XOR with the fix value 0x1B

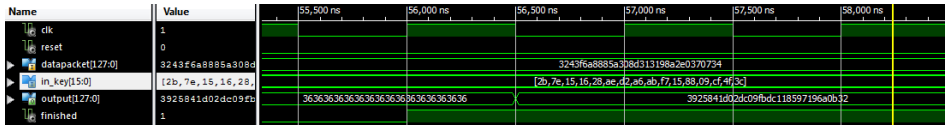


Figure 6.2: Test vector 1

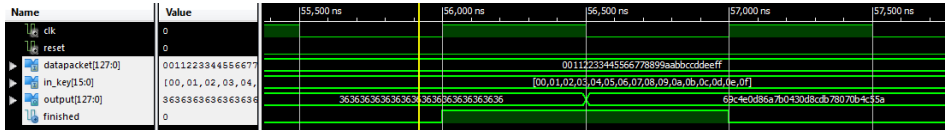


Figure 6.3: Test vector 2

if the shifted value exceeds 0xFF. A multiplication with 3 is the same as a multiplication with 2, followed by an XOR with the input value.

6.5 Tests

All of the entities in the design have been simulated and evaluated separately before being merged and tested together, to make sure that they had the desired functionality both separately and when combined together. The simulations for the separate blocks are trivial, and therefore not included in the report.

Figure 6.2 through 6.4 are tests performed on the complete aes-128 block, before CBC-mode. In the figures, `in_key` is the input key to be extended and used, and `datapacket` is one packet from a TS. Test vector 1 and 2 are taken from [NIST, 2001], while test vector 3 is generated using a webpage.

Test vector 1 (Figure 6.2)

Input key: 2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c

Plaintext: 32 43 f6 a8 88 5a 30 8d 31 31 98 a2 e0 37 07 34

Ciphertext: 39 25 84 1d 02 dc 09 fb dc 11 85 97 19 6a 0b 32

Test vector 2 (Figure 6.3)

Input key: 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f

Plaintext: 00 11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff

Ciphertext: 69 c4 e0 d8 6a 7b 04 30 d8 cd b7 80 70 b4 c5 5a

Test vector 3 (Figure 6.4)

Input key: 10 20 30 40 50 60 70 80 90 a0 b0 c0 d0 e0 f0 bb

Plaintext: 00 11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff

Ciphertext: bf 99 1f aa 8b 0f e6 48 36 46 a0 2d 33 9e de a5

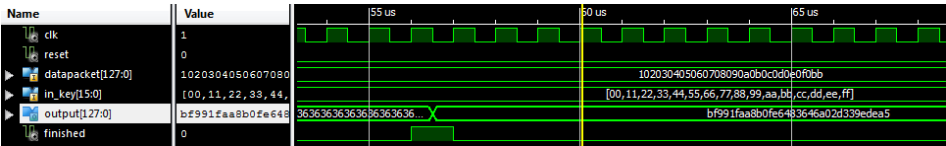


Figure 6.4: Test vector 3

Appendix

A

Matrixes

<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

$Rcon[256] = \{$

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.4: The Rcon function represented as a vector

B

Illustrations

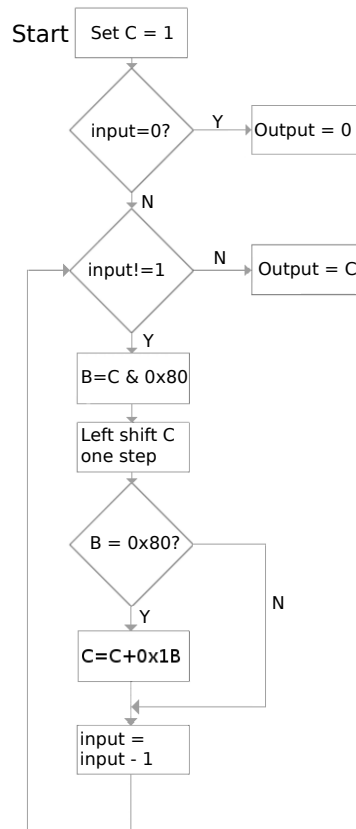


Figure B.1: Flowchart of the Rcon function

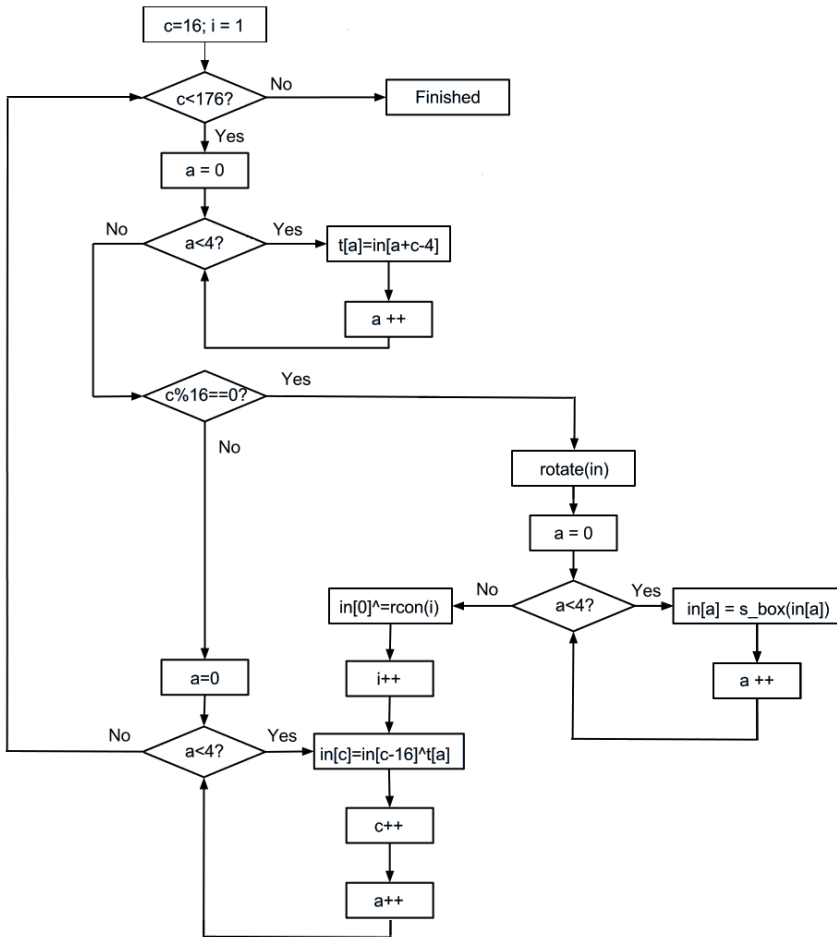


Figure B.2: Flowchart of the key schedule

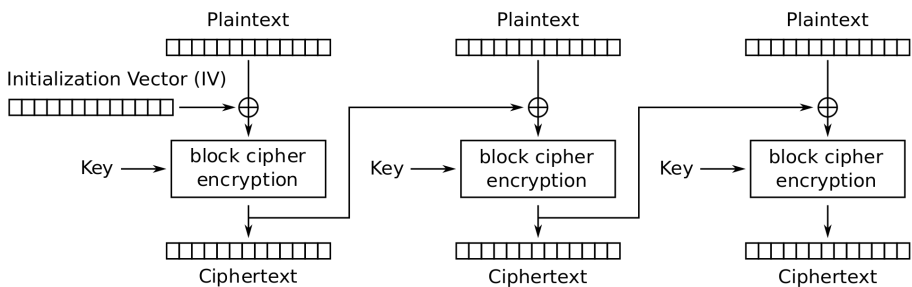


Figure B.3: Cipher block chaining mode, [Wikipedia, 2014a]

C

Test vectors

C.1 Test cases

This section contains test cases, which can be followed one step at the time.

The following test case is taken from NIST [2001, pp. 35–36]. The plaintext is input into a single aes-128 cipher.

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  3CAA3E8A99F9DEB50F3AF57ADF622AA
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

```

Table C.1 displays a keyexpansion based on a test case taken from NIST [2001, pp. 35–36].

Key = 2B 7E 15 16 28 AE D2 A6 AB F7 15 88 09 CF 4F 3C

i(dec)	temp	After RotWord	After SubWord	Rcon(i)	After \oplus with Rcon	w[i-16]	w[i] = temp \oplus w[i - 16]
4	09cf4f3c	cf4f3c09	8a84eb01	01000000	8b84eb01	2b7e1516	a0fafa17
5	a0fafa17					28aed2a6	88542cb1
6	88542cb1					abf71588	23a33939

7	23a33939					09cf4f3c	2a6c7605
8	2a6c7605	6c76052a	50386be5	02000000	52386be5	a0fafe17	f2c295f2
9	f2c295f2					88542cb1	7a96b943
10	7a96b943					23a33939	5935807a
11	5935807a					2a6c7605	7359f67f
12	7359f67f	59f67f73	cb42d28f	04000000	cf42d28f	f2c295f2	3d80477d
13	3d80477d					7a96b943	4716fe3e
14	4716fe3e					5935807a	1e237e44
15	1e237e44					7359f67f	6d7a883b
16	6d7a883b	7a883b6d	dac4e23c	08000000	d2c4e23c	3d80477d	ef44a541
17	ef44a541					4716fe3e	a8525b7f
18	a8525b7f					1e237e44	b671253b
19	b671253b					6d7a883b	db0bad00
20	db0bad00	0bad00db	2b9563b9	10000000	3b9563b9	ef44a541	d4d1c6f8
21	d4d1c6f8					a8525b7f	7c839d87
22	7c839d87					b671253b	caf2b8bc
23	caf2b8bc					db0bad00	11f915bc
24	11f915bc	f915bc11	99596582	20000000	b9596582	d4d1c6f8	6d88a37a
25	6d88a37a					7c839d87	110b3efd
26	110b3efd					caf2b8bc	dbf98641
27	dbf98641					11f915bc	ca0093fd
28	ca0093fd	0093fdca	63dc5474	40000000	23dc5474	6d88a37a	4e54f70e
29	4e54f70e					110b3efd	5f5fc9f3
30	5f5fc9f3					dbf98641	84a64fb2
31	84a64fb2					ca0093fd	4ea6dc4f
32	4ea6dc4f	a6dc4f4e	2486842f	80000000	a486842f	4e54f70e	ead27321
33	ead27321					5f5fc9f3	b58dbad2
34	b58dbad2					84a64fb2	312bf560
35	312bf560					4ea6dc4f	7f8d292f
36	7f8d292f	8d292f7f	5da515d2	1b000000	46a515d2	ead27321	ac7766f3
37	ac7766f3					b58dbad2	19fadc21
38	19fadc21					312bf560	28d12941
39	28d12941					7f8d292f	575c006e
40	575c006e	5c006e57	4a639f5b	36000000	7c639f5b	ac7766f3	d014f9a8
41	d014f9a8					19fadc21	c9ee2589
42	c9ee2589					28d12941	e13f0cc8
43	e13f0cc8					575c006e	b6630ca6

Table C.1: Keyexpansion

Table C.2 is a test case taken from NIST [2001, pp. 35–36].
16 bytes of data are run on a single aes-128 cipher.

Plaintext: 32 43 f6 a8 88 5a 30 8d 31 31 98 a2 e0 32 07 34
Key: 2B 7E 15 16 28 AE D2 A6 AB F7 15 88 09 CF 4F 3C

Round	Start of	After	After	After	Round Key
-------	----------	-------	-------	-------	-----------

Number	Round	SubBytes	ShiftRows	MixColumns		Value
input	328831e0				\oplus	2b28ab09
	435a3137					7eae f7cf
	f6309807					15d2154f
	a88da234					16a6883c
1	19a09ae9	d4e0b81e	d4e0b81e	04e04828	\oplus	a088232a
	3df4c6f8	27bf b441	bf b44127	66cbf806		fa54a36c
	e3e28d48	11985d52	5d521198	8119d326		fe2c3976
	be2b2a08	ae f1e530	30ae f1e5	e59a7a4c		17b13905
2	a4686b02	49457f77	49457f77	581bdb1b	\oplus	f27a5973
	9c9f5b6a	dedb3902	db3902de	4d4be76b		c2963559
	7f35ea50	d2968753	8753d296	ca5aca b0		95b980f6
	f22b4349	89f11a3b	3b89f11a	f1acae8e5		f2437a7f
3	aa618268	ac ef1345	ac ef1345	752053bb	\oplus	3d471e6d
	8fdd d232	73c1 b523	c1b52373	ec0bc025		8016237a
	5fe34a46	cf11 d65a	d65acf11	0963cf d0		47fe7e88
	03ef d29a	7bdf b5b8	b87bdf b5	93337c dc		7d3e443b
4	48674dd6	5285e3f6	5285e3f6	0f606f5e	\oplus	ef a8b6db
	6c1de35f	50a411cf	a411cf50	d631c0b3		4452710b
	4e9db158	2f5ec86a	c86a2f5e	da381013		a55b25ad
	ee0d38e7	28d70794	9428d707	a9bfb6b01		4a7f3b00
5	e0c8d985	e1e83597	e1e83597	25bdb64c	\oplus	d47c ca11
	9263b1b8	4ffbc86c	fb c86c4f	d1113a4c		d18df2f9
	7f6335be	d2fb96ae	96aed2fb	a9d133c0		c69db815
	e8c05001	9bba537c	7c9bba53	ad688eb0		f887bc bc
6	f1c17c5d	a178104c	a178104c	4b2c3337	\oplus	6d11db ca
	0092c8b5	634f e8d5	4f e8d563	864a9dd2		880bf900
	6f4c8bd5	a8293d03	3d03a829	8d89f418		a33e8693
	55ef320c	fcdf23fe	fe fcd f23	6d80e8d8		7afd41fd
7	263de8fd	f7279b54	f7279b54	14462734	\oplus	4e5f844e
	0e4164d2	ab8343b5	8343b5ab	1516462a		545fa6a6
	2eb7728b	31a9403d	403d31a9	b51556d8		f7c94f dc
	177da925	f0ff d33f	3ff0ff d3	bfecd743		0ef3b24f
8	5a19a37a	bed40ada	bed40ada	00b154fa	\oplus	ea b5317f
	4149e08c	833be164	3be16483	51c8761b		d28d2b8d
	42dc1904	2c86d4f2	d4f22c86	2f896d99		73ba f529
	b11f650c	c8c04dfe	fec8c04d	d1ffcd ea		21d2602f

9	<table><tr><td>ea</td><td>04</td><td>65</td><td>85</td></tr><tr><td>83</td><td>45</td><td>5d</td><td>96</td></tr><tr><td>5c</td><td>33</td><td>98</td><td>b0</td></tr><tr><td>f0</td><td>2d</td><td>ad</td><td>c5</td></tr></table>	ea	04	65	85	83	45	5d	96	5c	33	98	b0	f0	2d	ad	c5	<table><tr><td>87</td><td>f2</td><td>4d</td><td>97</td></tr><tr><td>ec</td><td>6e</td><td>4c</td><td>90</td></tr><tr><td>4a</td><td>c3</td><td>46</td><td>e7</td></tr><tr><td>8c</td><td>d8</td><td>95</td><td>a6</td></tr></table>	87	f2	4d	97	ec	6e	4c	90	4a	c3	46	e7	8c	d8	95	a6	<table><tr><td>87</td><td>f2</td><td>4d</td><td>97</td></tr><tr><td>6e</td><td>4c</td><td>90</td><td>ec</td></tr><tr><td>46</td><td>e7</td><td>4a</td><td>c3</td></tr><tr><td>a6</td><td>8c</td><td>d8</td><td>95</td></tr></table>	87	f2	4d	97	6e	4c	90	ec	46	e7	4a	c3	a6	8c	d8	95	<table><tr><td>47</td><td>40</td><td>a3</td><td>4c</td></tr><tr><td>37</td><td>d4</td><td>70</td><td>9f</td></tr><tr><td>94</td><td>e4</td><td>3a</td><td>42</td></tr><tr><td>ed</td><td>a5</td><td>a6</td><td>bc</td></tr></table>	47	40	a3	4c	37	d4	70	9f	94	e4	3a	42	ed	a5	a6	bc	\oplus	<table><tr><td>ac</td><td>19</td><td>28</td><td>57</td></tr><tr><td>77</td><td>fa</td><td>d1</td><td>5c</td></tr><tr><td>66</td><td>dc</td><td>29</td><td>00</td></tr><tr><td>f3</td><td>21</td><td>41</td><td>6e</td></tr></table>	ac	19	28	57	77	fa	d1	5c	66	dc	29	00	f3	21	41	6e
	ea	04	65	85																																																																																		
	83	45	5d	96																																																																																		
	5c	33	98	b0																																																																																		
f0	2d	ad	c5																																																																																			
87	f2	4d	97																																																																																			
ec	6e	4c	90																																																																																			
4a	c3	46	e7																																																																																			
8c	d8	95	a6																																																																																			
87	f2	4d	97																																																																																			
6e	4c	90	ec																																																																																			
46	e7	4a	c3																																																																																			
a6	8c	d8	95																																																																																			
47	40	a3	4c																																																																																			
37	d4	70	9f																																																																																			
94	e4	3a	42																																																																																			
ed	a5	a6	bc																																																																																			
ac	19	28	57																																																																																			
77	fa	d1	5c																																																																																			
66	dc	29	00																																																																																			
f3	21	41	6e																																																																																			
10	<table><tr><td>eb</td><td>59</td><td>8b</td><td>1b</td></tr><tr><td>40</td><td>2e</td><td>a1</td><td>c3</td></tr><tr><td>f2</td><td>38</td><td>13</td><td>42</td></tr><tr><td>1e</td><td>84</td><td>e7</td><td>d2</td></tr></table>	eb	59	8b	1b	40	2e	a1	c3	f2	38	13	42	1e	84	e7	d2	<table><tr><td>e9</td><td>cb</td><td>3d</td><td>af</td></tr><tr><td>09</td><td>31</td><td>32</td><td>e2</td></tr><tr><td>89</td><td>07</td><td>7d</td><td>2c</td></tr><tr><td>72</td><td>5f</td><td>94</td><td>b5</td></tr></table>	e9	cb	3d	af	09	31	32	e2	89	07	7d	2c	72	5f	94	b5	<table><tr><td>e9</td><td>cb</td><td>3d</td><td>af</td></tr><tr><td>31</td><td>32</td><td>e2</td><td>09</td></tr><tr><td>7d</td><td>2c</td><td>89</td><td>07</td></tr><tr><td>b5</td><td>72</td><td>5f</td><td>94</td></tr></table>	e9	cb	3d	af	31	32	e2	09	7d	2c	89	07	b5	72	5f	94	<table><tr><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td></tr></table>																	\oplus	<table><tr><td>d0</td><td>c9</td><td>e1</td><td>b6</td></tr><tr><td>14</td><td>ee</td><td>3f</td><td>63</td></tr><tr><td>f9</td><td>25</td><td>0c</td><td>0c</td></tr><tr><td>a8</td><td>89</td><td>c8</td><td>a6</td></tr></table>	d0	c9	e1	b6	14	ee	3f	63	f9	25	0c	0c	a8	89	c8	a6
	eb	59	8b	1b																																																																																		
	40	2e	a1	c3																																																																																		
	f2	38	13	42																																																																																		
1e	84	e7	d2																																																																																			
e9	cb	3d	af																																																																																			
09	31	32	e2																																																																																			
89	07	7d	2c																																																																																			
72	5f	94	b5																																																																																			
e9	cb	3d	af																																																																																			
31	32	e2	09																																																																																			
7d	2c	89	07																																																																																			
b5	72	5f	94																																																																																			
d0	c9	e1	b6																																																																																			
14	ee	3f	63																																																																																			
f9	25	0c	0c																																																																																			
a8	89	c8	a6																																																																																			
output	<table><tr><td>39</td><td>02</td><td>dc</td><td>19</td></tr><tr><td>25</td><td>dc</td><td>11</td><td>6a</td></tr><tr><td>84</td><td>09</td><td>85</td><td>0b</td></tr><tr><td>1d</td><td>fb</td><td>97</td><td>32</td></tr></table>	39	02	dc	19	25	dc	11	6a	84	09	85	0b	1d	fb	97	32																																																																					
39	02	dc	19																																																																																			
25	dc	11	6a																																																																																			
84	09	85	0b																																																																																			
1d	fb	97	32																																																																																			

Table C.2: AES-128 Scrambling on 16 byte packet

Table C.3 is a test case for the CBC-mode scrambling performed on a TS-packet. The highlighted bytes are left in the clear, due to the scrambling only working with packets consisting of 16 bytes.

Clear Packet	<table><tr><td>47 60 80 11</td><td>54 68 69 73 20 69 73 20 74 68 65 20 70 61 79 6C 6F 61 64 20 75 73 65 64 20 66 6F 72 20 63 72 65 61 74 69 6E 67 20 74 68 65 20 74 65 73 74 20 76 65 63 74 6F 72 73 20 66 6F 72 20 74 68 65 20 44 56 42 20 49 50 54 56 20 73 63 72 61 6D 62 6C 65 72 2F 64 65 73 63 72 61 6D 62 6C 65 72 7E 20 54 68 69 73 20 69 73 20 74 68 65 20 70 61 79 6C 6F 61 64 20 75 73 65 64 20 66 6F 72 20 63 72 65 61 74 69 6E 67 20 74 68 65 20 74 65 73 74 20 76 65 63 74 6F 72 73 20 66 6F 72 20 74 68 65 20 44 56 42 20 49 50 54 56 20 73 63 72 61 6D 62 6C 65 72 2F 64 65 73 63 72 61 6D</td></tr></table>	47 60 80 11	54 68 69 73 20 69 73 20 74 68 65 20 70 61 79 6C 6F 61 64 20 75 73 65 64 20 66 6F 72 20 63 72 65 61 74 69 6E 67 20 74 68 65 20 74 65 73 74 20 76 65 63 74 6F 72 73 20 66 6F 72 20 74 68 65 20 44 56 42 20 49 50 54 56 20 73 63 72 61 6D 62 6C 65 72 2F 64 65 73 63 72 61 6D 62 6C 65 72 7E 20 54 68 69 73 20 69 73 20 74 68 65 20 70 61 79 6C 6F 61 64 20 75 73 65 64 20 66 6F 72 20 63 72 65 61 74 69 6E 67 20 74 68 65 20 74 65 73 74 20 76 65 63 74 6F 72 73 20 66 6F 72 20 74 68 65 20 44 56 42 20 49 50 54 56 20 73 63 72 61 6D 62 6C 65 72 2F 64 65 73 63 72 61 6D
47 60 80 11	54 68 69 73 20 69 73 20 74 68 65 20 70 61 79 6C 6F 61 64 20 75 73 65 64 20 66 6F 72 20 63 72 65 61 74 69 6E 67 20 74 68 65 20 74 65 73 74 20 76 65 63 74 6F 72 73 20 66 6F 72 20 74 68 65 20 44 56 42 20 49 50 54 56 20 73 63 72 61 6D 62 6C 65 72 2F 64 65 73 63 72 61 6D 62 6C 65 72 7E 20 54 68 69 73 20 69 73 20 74 68 65 20 70 61 79 6C 6F 61 64 20 75 73 65 64 20 66 6F 72 20 63 72 65 61 74 69 6E 67 20 74 68 65 20 74 65 73 74 20 76 65 63 74 6F 72 73 20 66 6F 72 20 74 68 65 20 44 56 42 20 49 50 54 56 20 73 63 72 61 6D 62 6C 65 72 2F 64 65 73 63 72 61 6D		
Scrambled Packet	<table><tr><td>47 60 80 11</td><td>15 CE 67 E0 CB 01 B5 3C E7 60 54 E5 7A 4A D1 20 A0 DF A4 EA AA E9 32 C6 78 3F 51 AE 19 FA EE 10 8B DB 78 F3 11 3E C2 B5 72 CC 20 85 00 A5 2C EC A1 14 12 6C 58 24 4D F5 63 E7 A9 B4 E0 41 CB C3 FB FF FB D8 3C 8F BF FB 10 E8 3E A3 82 04 BA D7 02 FB 01 A2 7B 62 2C 4F 85 AA B6 AA 75 55 97 20 D6 5A B8 44 CE A2 8C F2 E1 FE 5E 7A C1 9D 44 81 89 19 C2 32 49 F1 40 75 7B 5D 16 C0 AF 45 B2 5F 50 9B 9D A0 61 97 12 C5 9F 0B 30 B0 6F 1F BE 90 12 3F 21 29 83 93 6A 95 31 7F CB 62 F4 34 6A 1B 1E 16 48 40 30 3A FF 83 8A 01 9B F8</td></tr></table>	47 60 80 11	15 CE 67 E0 CB 01 B5 3C E7 60 54 E5 7A 4A D1 20 A0 DF A4 EA AA E9 32 C6 78 3F 51 AE 19 FA EE 10 8B DB 78 F3 11 3E C2 B5 72 CC 20 85 00 A5 2C EC A1 14 12 6C 58 24 4D F5 63 E7 A9 B4 E0 41 CB C3 FB FF FB D8 3C 8F BF FB 10 E8 3E A3 82 04 BA D7 02 FB 01 A2 7B 62 2C 4F 85 AA B6 AA 75 55 97 20 D6 5A B8 44 CE A2 8C F2 E1 FE 5E 7A C1 9D 44 81 89 19 C2 32 49 F1 40 75 7B 5D 16 C0 AF 45 B2 5F 50 9B 9D A0 61 97 12 C5 9F 0B 30 B0 6F 1F BE 90 12 3F 21 29 83 93 6A 95 31 7F CB 62 F4 34 6A 1B 1E 16 48 40 30 3A FF 83 8A 01 9B F8
47 60 80 11	15 CE 67 E0 CB 01 B5 3C E7 60 54 E5 7A 4A D1 20 A0 DF A4 EA AA E9 32 C6 78 3F 51 AE 19 FA EE 10 8B DB 78 F3 11 3E C2 B5 72 CC 20 85 00 A5 2C EC A1 14 12 6C 58 24 4D F5 63 E7 A9 B4 E0 41 CB C3 FB FF FB D8 3C 8F BF FB 10 E8 3E A3 82 04 BA D7 02 FB 01 A2 7B 62 2C 4F 85 AA B6 AA 75 55 97 20 D6 5A B8 44 CE A2 8C F2 E1 FE 5E 7A C1 9D 44 81 89 19 C2 32 49 F1 40 75 7B 5D 16 C0 AF 45 B2 5F 50 9B 9D A0 61 97 12 C5 9F 0B 30 B0 6F 1F BE 90 12 3F 21 29 83 93 6A 95 31 7F CB 62 F4 34 6A 1B 1E 16 48 40 30 3A FF 83 8A 01 9B F8		

	10 A8 E0 B2 2F 64 65 73 63 72 6A 6D
--	-------------------------------------

Table C.3: TS packet scrambled in cbc-mode

C.2 Keyexpansion

This section only contains input keys, and the respective expanded keys.

Input key: 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Output key:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
	62	63	63	63	62	63	63	63	62	63	63	63	62	63	63
	9b	98	98	c9	f9	fb	fb	aa	9b	98	98	c9	f9	fb	aa
	90	97	34	50	69	6c	cf	fa	f2	f4	57	33	0b	0f	ac
	ee	06	da	7b	87	6a	15	81	75	9e	42	b2	7e	91	ee
	7f	2e	2b	88	f8	44	3e	09	8d	da	7c	bb	f3	4b	92
	ec	61	4b	85	14	25	75	8c	99	ff	09	37	6a	b4	a7
	21	75	17	87	35	50	62	0b	ac	af	6b	3c	c6	1b	f0
	0e	f9	03	33	3b	a9	61	38	97	06	0a	04	51	1d	fa
	b1	d4	d8	e2	8a	7d	b9	da	1d	7b	b3	de	4c	66	49
	b4	ef	5b	cb	3e	92	e2	11	23	e9	51	cf	6f	8f	18

Input key : ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff

Output key:	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff
	e8	e9	e9	e9	17	16	16	16	e8	e9	e9	e9	17	16	16
	ad	ae	ae	19	ba	b8	b8	0f	52	51	51	e6	45	47	f0
	09	0e	22	77	b3	b6	9a	78	e1	e7	cb	9e	a4	a0	8c
	e1	6a	bd	3e	52	dc	27	46	b3	3b	ec	d8	17	9b	60
	e5	ba	f3	ce	b7	66	d4	88	04	5d	38	50	13	c6	58
	71	d0	7d	b3	c6	b6	a9	3b	c2	eb	91	6b	d1	2d	c9
	e9	0d	20	8d	2f	bb	89	b6	ed	50	18	dd	3c	7d	d1
	96	33	73	66	b9	88	fa	d0	54	d8	e2	0d	68	a5	33
	8b	f0	3f	23	32	78	c5	f3	66	a0	27	fe	0e	05	14
	d6	0a	35	88	e4	72	f0	7b	82	d2	d7	85	8c	d7	c3

Input key : 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f

Output key:	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e
	d6	aa	74	fd	d2	af	72	fa	da	a6	78	f1	d6	ab	76
	b6	92	cf	0b	64	3d	bd	f1	be	9b	c5	00	68	30	b3
	b6	ff	74	4e	d2	c2	c9	bf	6c	59	0c	bf	04	69	bf
	47	f7	f7	bc	95	35	3e	03	f9	6c	32	bc	fd	05	8d
	3c	aa	a3	e8	a9	9f	9d	eb	50	f3	af	57	ad	f6	22
	5e	39	0f	7d	f7	a6	92	96	a7	55	3d	c1	0a	a3	1f
	14	f9	70	1a	e3	5f	e2	8c	44	0a	df	4d	4e	a9	c0
	47	43	87	35	a4	1c	65	b9	e0	16	ba	f4	ae	bf	7a
	54	99	32	d1	f0	85	57	68	10	93	ed	9c	be	2c	97

13 11 1d 7f e3 94 4a 17 f3 07 a7 8b 4d 2b 30 c5

Input key : 00 11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff

Output key:	00	11	22	33	44	55	66	77	88	99	aa	bb	cc	dd	ee	ff
	c0	39	34	78	84	6c	52	0f	0c	f5	f8	b4	c0	28	16	4b
	f6	7e	87	c2	72	12	d6	cd	7e	e7	2d	79	be	cf	3b	32
	78	9c	a4	6c	0a	8e	71	a1	74	69	5c	d8	ca	a6	67	ea
	54	19	23	18	5e	97	52	b9	2a	fe	0e	61	e0	58	69	8b
	2e	e0	1e	f9	70	77	4c	40	5a	89	42	21	ba	d1	2b	aa
	30	11	b2	0d	40	66	fe	4d	1a	ef	bc	6c	a0	3e	97	c6
	c2	99	06	ed	82	ff	f8	a0	98	10	44	cc	38	2e	d3	0a
	73	ff	61	ea	f1	00	99	4a	69	10	dd	86	51	3e	0e	8c
	da	54	05	3b	2b	52	9c	71	42	44	41	f7	13	7a	4f	7b
	36	d0	24	46	1d	84	b8	37	5f	c0	f9	c0	4c	ba	b6	bl

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

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

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

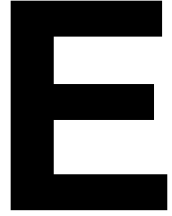
The inverse of the XOR-operation the XOR-operation.

This gives us:

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

which gives us

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



Layout of the circuit

This purpose of this chapter is to give an overview of what the circuit is supposed to look like, realized using blocks. The top entity is the `aes_scrambler` (Figure E.1). Note that the manager-entity in Figure E.1 and E.5 are not the same entities.

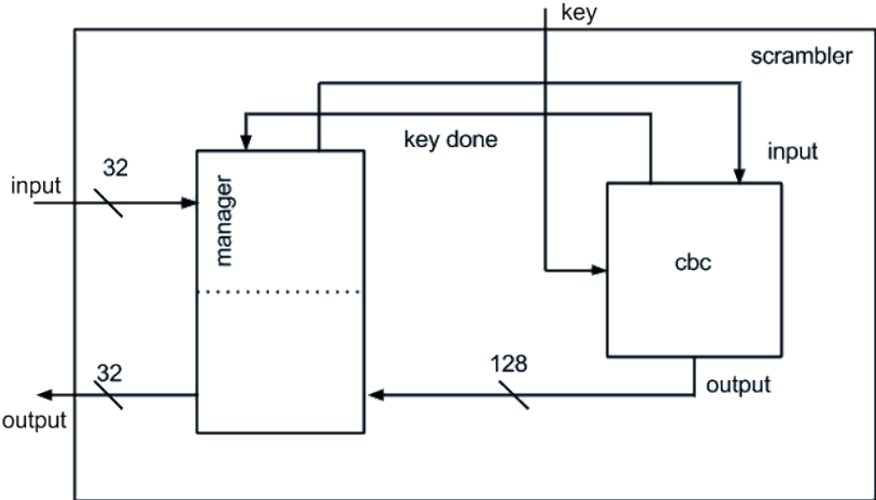


Figure E.1: Scrambler-block

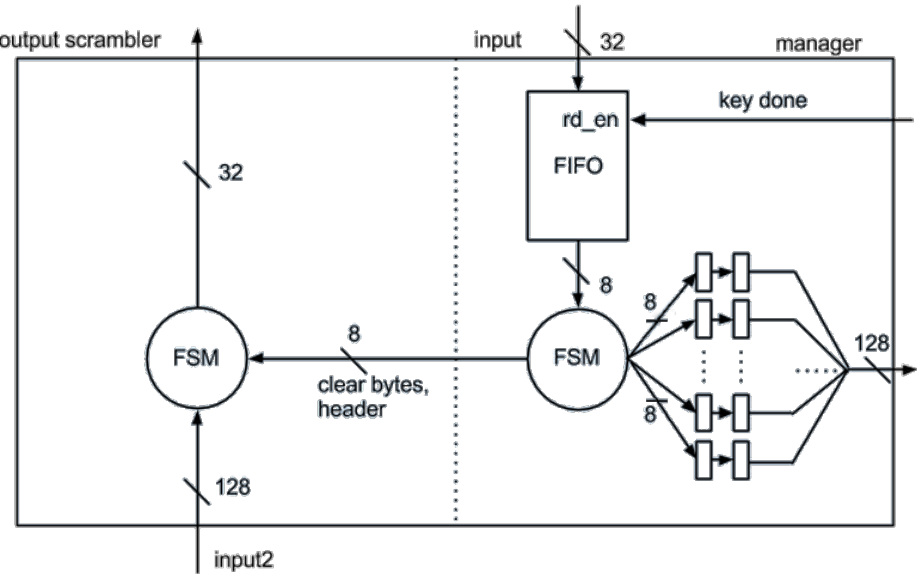


Figure E.2: Manager-block

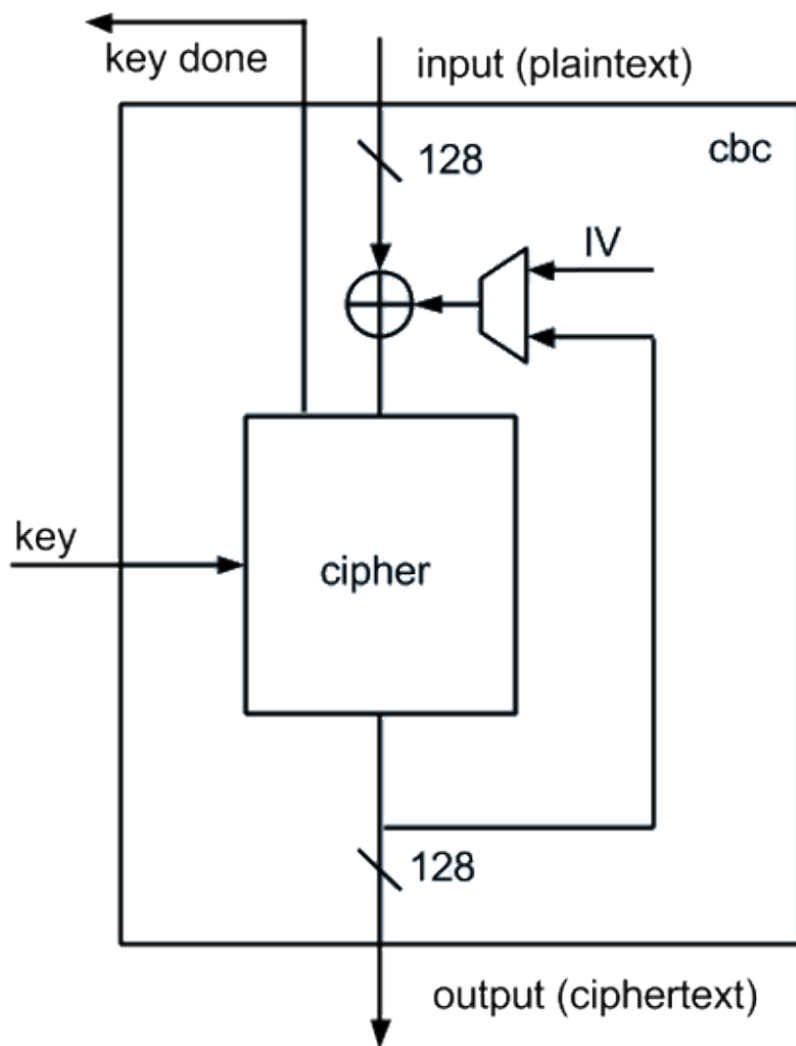


Figure E.3: CBC-block

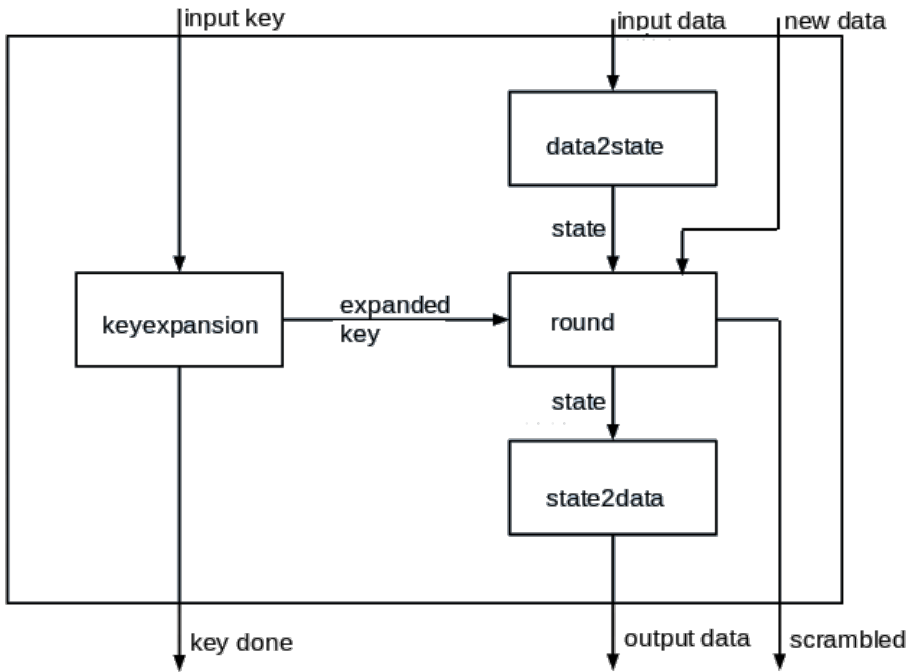


Figure E.4: Cipher-block

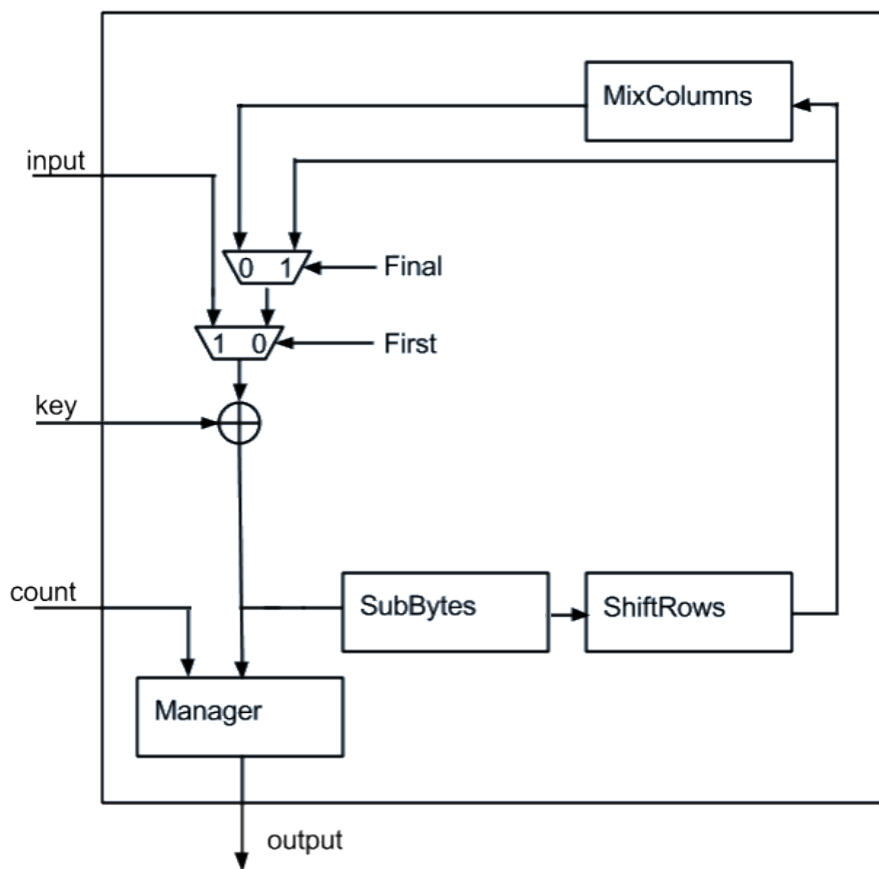


Figure E.5: Round-block

List of Figures

1.1	CIPlus interface. Image remade from [LLP, 2011, p. 10]	3
2.1	General layout of a data packet	6
2.2	PES packet derived from TS packets	8
2.3	Different kinds of ciphers [Wikipedia, 2014b]	9
2.4	SP-Network	11
3.1	Number of bits in key used	14
6.1	The top entity	27
6.2	Test vector 1	30
6.3	Test vector 2	30
6.4	Test vector 3	31
A.1	Rijndael S-box	36
A.2	State-Matrix	36
A.3	Rijndael MixColumns equation	36
A.4	The Rcon function represented as a vector	37
B.1	Flowchart of the Rcon function	40
B.2	Flowchart of the key schedule	41
B.3	Cipher block chaining mode, [Wikipedia, 2014a]	41
E.1	Scrambler-block	54
E.2	Manager-block	54
E.3	CBC-block	55
E.4	Cipher-block	56
E.5	Round-block	57

Bibliography

- Conax, 2014. URL <http://www.conax.com/about-us/>. Accessed: 13 Feb 2014. Not cited.
- 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 8.
- 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 2.
- 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, 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://www.farncombe.co.uk/whitepaper/>.

- [//farncombe.eu/whitepapers/FTLCAWhitePaperTwo.pdf](http://farncombe.eu/whitepapers/FTLCAWhitePaperTwo.pdf). Accessed: 28 jan 2014. Cited on pages 13 and 14.
- 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. Security analysis of dvb common scrambling algorithm. In *Data, Privacy, and E-Commerce, 2007. ISDPE 2007. The First International Symposium on*, pages 271–273. IEEE, IEEE Xplore, 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-plus.com/data/ci-plus_overview_v2011-11-11.pdf. Accessed: 4 March 2014. Cited on pages 3 and 59.
- NIST. Specification for the advanced encryption standard (aes), November 2001. URL <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>. Accessed: 17 Feb 2014. Cited on pages 30, 43, 44, and 45.
- Bruce Schneier and Niels Ferguson. *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-ID_CWProtection_May_25.pdf. Accessed: 18 Feb, 2014. Cited on page 14.
- C. E. Shannon. Communication theory of secrecy systems*. *Bell System Technical Journal*, 28(4), 1949. ISSN 1538-7305. doi: 10.1002/j.1538-7305.1949.tb00928.x. URL <http://dx.doi.org/10.1002/j.1538-7305.1949.tb00928.x>. 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 Simpson, Matt Henricksen, and Wun-She Yap. Improved cryptanalysis of the common scrambling algorithm stream cipher. In *Information Security and Privacy*, pages 108–121. Springer, 2009. URL <http://eprints.qut.edu.au/27578/1/c27578.pdf>. Accessed: 12 Feb 2014. 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. In *Research in Cryptology*, pages 45–61. Springer, 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 page 15.

- Serge Vaudenay, Willi Meier, Simon Fischer, et al. 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.
- Ralf-Philipp Weinmann and Kai Wirt. Analysis of the dvb common scrambling algorithm. In *Communications and Multimedia Security*, pages 195–207. Springer, 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 41 and 59.
- 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 pages 9 and 59.
- 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.

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/>