

# The AES encryption algorithm

A analysis of The Advanced Encryption Standard (AES)

Klass:

**NA20**

Handledare:

**Jimmy Nylén**

Författare:

**Gabriel Lindeblad**

Program:

**Naturvetenskapsprogrammet**

# **Abstract**

This paper is a study of the Advanced Encryption Standard with the purpose of investigating how the key length and running mode affects the encryption speed of the algorithm as well as how the different running modes affect the security. The study was conducted by implementing AES and the running modes ECB, CBC and OFB in the programming language Python and then performing a series of tests on it. These tests consisted of measuring the encryption speed of the algorithm with different key lengths and running modes as well as the encryption of a image file using each of the running modes. The result of the study showed that the encryption speed of the algorithm increased as the key length increased, while a noticeable speed difference between the different running modes not could be assessed from the result of the study. The result of the study also showed that the running mode ECB was the least secure of the three running modes tested.

# Innehåll

<b>Ordlista</b>	<b>5</b>
<b>Akronymer</b>	<b>10</b>
<b>1 Inledning</b>	<b>11</b>
1.1 Syfte . . . . .	11
1.2 Frågeställningar . . . . .	11
1.3 Avgränsning . . . . .	11
<b>2 Bakgrund</b>	<b>12</b>
2.1 Kryptografi . . . . .	12
2.2 Varför behövs kryptering? . . . . .	12
2.2.1 Kryptografins uppkomst . . . . .	12
2.2.2 Kryptografins utveckling . . . . .	13
2.3 AES Uppkomst . . . . .	13
<b>3 Teori</b>	<b>14</b>
3.1 Kryptering . . . . .	14
3.2 Blockskiffer . . . . .	14
3.2.1 Körlägen . . . . .	14
3.2.1.1 ECB . . . . .	15
3.2.1.2 CBC . . . . .	16
3.2.1.3 OFB . . . . .	18
3.3 Symetrisk & Asymmetrisk Kryptering . . . . .	19
3.4 AES . . . . .	19
3.4.1 Finite Fields . . . . .	20
3.4.2 AES S-Box . . . . .	20
3.4.3 Struktur . . . . .	21
3.4.3.1 SubBytes-operationen . . . . .	24
3.4.3.2 ShiftRows operationen . . . . .	25
3.4.3.3 Inverse ShiftRows-operationen . . . . .	25
3.4.3.4 MixColumns-operationen . . . . .	26
3.4.3.5 Inverse MixColumns-operationen . . . . .	26
3.4.3.6 AddRoundKey-operationen . . . . .	27
3.4.4 Nyckelutökning . . . . .	28
3.4.4.1 RotWord . . . . .	29
3.4.4.2 SubWord . . . . .	29
3.4.4.3 Rcon . . . . .	29
<b>4 Metod &amp; Genomförande</b>	<b>31</b>
4.1 Implementering . . . . .	31
4.2 Testuppsättning . . . . .	31
4.2.1 Testmiljö . . . . .	31
4.2.2 Nyckellängdstest . . . . .	32
4.2.3 Körlägestest . . . . .	32
4.2.4 Krypteringstest . . . . .	32
4.3 Genomförande . . . . .	33

<b>5 Resultat</b>	<b>34</b>
5.1 Nyckellängdstest . . . . .	34
5.2 Körlägestest . . . . .	34
5.3 Krypteringstest . . . . .	34
<b>6 Diskussion</b>	<b>35</b>
6.1 Felkällor . . . . .	36
6.2 Förbättringar . . . . .	36
6.3 Slutsats . . . . .	37
<b>Källförteckning</b>	<b>39</b>
<b>A AES körlägestest resultat</b>	<b>42</b>
A.1 Före testet . . . . .	42
A.2 Efter ECB kryptering . . . . .	43
A.3 Efter CBC kryptering . . . . .	43
A.4 Efter CFB kryptering . . . . .	44
<b>B Fullständig data från testerna</b>	<b>45</b>
B.1 Data från nyckellängdstest . . . . .	45
B.2 Data från körlägestest . . . . .	46
<b>C AES Implementering i python</b>	<b>47</b>
C.1 AES.py . . . . .	47
C.2 encrypt.py . . . . .	59
C.3 decrypt.py . . . . .	59
C.4 __main__.py . . . . .	60
<b>D Test kod (Analyze.py)</b>	<b>63</b>

# Figur lista

3.1	Electronic Code Book läge kryptering . . . . .	15
3.2	Electronic Code Book läge dekryptering . . . . .	16
3.3	Cipher Block Chaining läge kryptering . . . . .	17
3.4	Cipher Block Chaining läge dekryptering . . . . .	17
3.5	Output Feedback läge kryptering . . . . .	18
3.6	Output Feedback läge dekryptering . . . . .	18
3.7	AES S-box (Den vänstra matrisen) & Invers S-box (Den högra matrisen) . . . . .	21
3.8	Vanlig runda . . . . .	22
3.9	Krypterings runda funktion . . . . .	23
3.10	Dekrypterings runda funktion . . . . .	24
3.11	SubBytes operationen . . . . .	24
3.12	Inverse SubBytes operationen . . . . .	25
3.13	ShiftRows operationen . . . . .	25
3.14	Inverse ShiftRows operationen . . . . .	25
3.15	MixColumns operationen . . . . .	26
3.16	Inverse MixColumns operationen . . . . .	27
3.17	AddRoundKey operationen . . . . .	27
3.18	Nyckel utöknings processen . . . . .	28
3.19	RotWord operationen . . . . .	29
3.20	SubWord operationen . . . . .	29
3.21	Rcon operationen . . . . .	30
5.1	AES krypterings test (ECB, CBC, OFB) . . . . .	34
A.1	Orginal bild . . . . .	42
A.2	Efter ECB Kryptering . . . . .	43
A.3	Efter CBC Kryptering . . . . .	43
A.4	Efter OFB Kryptering . . . . .	44

# Ordlista

A | B | C | E | F | G | H | M | N | O | P | R | S | V | W | X

## A

### AND

AND är en logisk operation inom datorvetenskap och matematik som tar två Binära värden och ger till baka ett Binärt värde. Detta värde är 1 om och endast om båda värdena är 1 annars är värdet 0.[[Wik22h](#)]

## B

### Binär

Binär är ett begrepp som används inom datorvetenskap och matematik för att beskriva ett värde som endast kan vara två olika saker som exempelvis 0 eller 1, sant eller falskt. Detta innebär att binära tal bygger på talbasen 2, vilket skiljer sig från vad som ofta vanligen används i vardagen som är talbasen 10.[[Wik21a](#)]

### Bit

En Bit är den minsta enheten av information som kan lagras i en dator. En bit kan endast ha två värden där den antingen är 0 eller 1, alltså ett Binärt värde. I datorvetenskap pratar man dock mer vanligen om ett Byte som är 8 bits.[[Wik22b](#)]

### Byte

En Byte består av 8 Bits och är en enhet som används inom datorvetenskap. En byte kan ha 256 olika värden från 0 till 255, vilket är  $2^8$  värden. Dessa värden representerar ofta tecken eller symboler som exempelvis bokstäver, siffror med mera. Tolkningen av vad en sekvens av bytes eller en enskild byte står för beror dock på vilken teckenkodning som används. Exempel på teckenkodningar kan vara ASCII och ISO-8859.[[Wik20](#)]

## C

### Caesarchiffer

Caesarchiffer är ett Substitutionsskiffer, vilket helt enkelt bygger på att man byter ut varje bokstav i meddelandet med en annan. Ersättningens bokstaven bestäms genom att man hoppar ett visst antal hopp i alfabetet som exempelvis 3 hopp, vilket då innebär att ifall man har bokstaven a då skulle den bli ett d istället.[[Wik21b](#)]

### CPU-klockhastighet

CPU-klockhastighet är den hastighet som en CPU kan utföra instruktioner på. Den mäts i hertz och är frekvensen som klockgeneratorn i en dators CPU använder sig av för att bland annat styra hur snabbt instruktioner ska utföras samt synka olika komponenter i datorn.[[Wik23a](#)]

## E

## **Enigma**

Enigma var ett krypterings verktyg som användes under andra världskriget av tyska militären för att kryptera meddelanden. Maskinen bestod av en elektromekanisk rotordisk som under tiden meddelandet skrivs in för kryptering även ändrar det elektriska kopplingarna mellan vilka bokstäver som blir vad. Detta är en av sakerna som gjorde Enigma väldigt svår att knäcka samt en av anledningarna till att liknande maskiner användes under stora delar av det tidiga 1900-talet.[[Wik22d](#)]

## **F**

### **Frekvensanalys**

Frekvensanalys inom kryptografi är en metod för att knäcka ett Substitutionsskiffer genom att analysera frekvensen av bokstäver och utnyttja det faktum att en del bokstäver framkommer mer frekvent än andra i språket. På detta viset kan man då sedan lista ut vilka bokstäver som är vilka i det krypterade meddelandet.[[Wik22f](#)]

## **G**

### **GIMP**

GIMP är ett open source bild redigerings program som är tillgängligt för det flesta operativsystem.[[Gim22](#)]

## **H**

### **Hashfunktion**

Hashfunktion är en funktion som delar upp en viss datamängd och genom för sedan en serie operationer som resulterar i en hashtext av godkänd längd. Längden är samma för alla hashtexter som använder samma funktion medan innehållet förändras så fort en enda Bit ändras i datamängden som funktionen appliceras på. Användningsområdet för dessa funktioner är bland annat när man vill verifiera meddelanden eller information och försäkra sig om att ingen ändrat på meddelandet efter att det skickats. Detta kan man då göra för att man vet att om man kör informationen genom samma hashfunktion borde resultatet vara identisk ifall informationen är oförändrad.[[Wik22g](#)]

### **Hexadecimal**

Hexadecimaltal är tal med talbasen 16. Detta innebär att det finns 16 olika symboler som kan användas istället för 10 symboler som används i decimaltal som har talbasen 10. Detta görs genom att man representerar tal större än 9 med bokstäverna A-F.[[Nat22a](#)]

## **HTTP**

HTTP (Hypertext Transfer Protocol) är det protokoll som används när man besöker en webbsida. Detta protokoll har använts sedan 1990-talet och används fortfarande till stora delar.[[BLFF96](#)]

## **M**

## Matrismultiplikation

Matrismultiplikation är en matematisk metod för att multiplicera ihop två eller flera matriser med varandra. Detta görs genom att värdena i den första matrisen multipliceras med värdena i den andra matrisen och sedan summeras ihop. Detta resulterar i en ny matris som är resultatet av multiplikationen.[[Abd19](#)]

## N

### NOT

NOT är en logisk operation inom datorvetenskap och matematik som tar två Binära värden och jämför dom. Det slutgiltiga värdet som ges tillbaka är 1 om värdena inte är lika varandra och annars är värdet 0.[[Wik22h](#)]

### Nyckelström

En nyckelström är i kryptografin en ström av Pseudoslump karakterer som kan kombineras med exempelvis ett meddelande för att producera en skiffrertext.[[Wik21c](#)]

## O

### Operativsystem

Operativsystem är det program på en dator som fungerar som gränssnitt mellan användaren och datorns maskinvara. Detta innebär att operativsystemet hanterar saker som inmatning från användaren till att tillhandahålla ett gränssnitt för olika program så att dom kan genomföra sina uppgifter.[[Wik22a](#)]

## OR

OR är en logisk operation inom datorvetenskap och matematik som tar två Binära värden och ger till baka ett Binärt värd. Detta värdet är 1 om minst ett av värdena är 1 annars är värdet 0.[[Wik22h](#)]

## P

### Polyalphabetic substitutionsskiffer

Polyalphabetic substitutionsskiffer bygger på att man använder flera olika Substitutionsskiffer för att på så sätt undvika en utav det största svagheterna med Substitutionsskiffer. Detta då att dom lätt går att knäcka genom en Frekvensanalys då vissa bokstäver dyker upp mer frekvent i språket än andra. För att lösa detta så använder polyalphabetiska skiffer flera olika substitutionsskiffer som man byter mellan med en viss frekvens för att eliminera Frekvensanalysens effektivitet.[[Wik22i](#)]

## PPM

PPM eller även kallad Portable Pixel Map är ett filformat som förvara bilder i råa bytes som då representerar färgerna i bilden. PPM använder sig av 3 Bytes för att representera en färg.[[Fil22](#)]

### Pseudoslump

Pseudoslump är en rad av nummer som kan se ut att vara helt slumpmässiga men har blivit framställda genom en upprepbar process.[[Wik22j](#)]

## Python

Python är ett högnivå programmeringsspråk som är skrivet i programmerings språket C. Det är skapat av Guido van Rossum och släpptes i februari 1991.[[Pyt22b](#)]

## R

### RSA

Rivest-Shamir-Adleman (RSA) är en av det mest välkända krypteringsalgoritmerna och var en av det första algoritmerna som byggde på en asymmetrisk kryptering. RSA bygger på multiplikation av stora primtal där primtalen är nycklarna.[[Wik22k](#)]

## S

### SP-network

SP-network eller även kallat Substitution-permutation network är inom kryptografin en serie av matematiska operationer som genomförs i rundor för att på så sätt kryptera ett meddelande. Det består ut av två delar, en substitutions del och en permutation del. Substitutionen delen fungerar precis som SubBytes-operationen medans permutationen exempelvis skulle kunna representeras med ShiftRows operationen.[[Wik22o](#)]

## SSH

SSH eller som det även är kallat Secure Shell är ett protokoll som används för säker kommunikation över ett nätverk. SSH är ett exempel på ett nätverksprotokoll och används bland annat för att kryptera meddelanden som skickas mellan två enheter.[[Bar+01](#)]

### Strömskiffer

Strömskiffer, ett symmetriskt nyckelskiffer där man använder en Pseudoslumpmässig skiffrerström (Nyckelström) som sedan en Bit i taget kombineras med det som ska krypteras. Den kombinerande operationen som används i strömskiffer är ofta en XOR-operation.[[Wik22m](#)]

### Substitutionsskiffer

Ett Substitutionsskiffer är en typ av krypteringsteknik som bygger på att man byter ut delar av informationen man ska kryptera med exempelvis andra symboler med hjälp av en nyckel. Detta kan exempelvis vara bokstäver som byts ut mot andra bokstäver precis som i Caesarchiffer eller siffror som byts ut mot andra siffror eller bokstäver.[[Wik22n](#)]

## V

### VSCode

Visual Studio Code är en programutvecklingsmiljö som är skapad av Microsoft. Det är ett öppet källkods projekt som är tillgängligt för det flesta operativsystem och kan användas för att skriva kod i flera olika språk.[[Wik21d](#)]

## W

### Windows 11

Windows 11 är den senaste versionen av ett Operativsystem för datorer framtaget av företaget Microsoft. Windows 11 tillhör Operativsystem familjen Windows NT som först lanserades 1993.[[Wik23b](#)]

**X**

## **XOR**

XOR är en logisk operation inom datorvetenskap som fungerar ungefär som  $+$  uttrycket, med den enda skillnaden att  $1 \oplus 1 = 0$ . Detta samt att XOR är en Binär operation, vilket innebär att termerna bara kan vara 0 eller 1 och resultatet det samma. Utöver XOR finns även OR, NOT och AND bland annat.[\[LEW12\]](#)

# Akronymer

<b>AES</b>	Advanced Encryption Standard
<b>ASCII</b>	American Standard Code for Information Interchange
<b>CBC</b>	Cipher Block Chaining läge
<b>DES</b>	Data Encryption Standard
<b>ECB</b>	Electronic Code Book läge
<b>IV</b>	Initialization Vector
<b>NIST</b>	National Institute of Standards and Technology
<b>OFB</b>	Output Feedback läge
<b>Rcon</b>	Round Constant
<b>SSL</b>	Secure Socket Layer
<b>TLS</b>	Transport Layer Security
<b>WPA2</b>	Wi-Fi Protected Access 2

# 1 Inledning

Kryptering, en bärande grundsten i dagens digitaliserade samhälle. Det är väggen mellan oss och resten av världen, ett läs runt våra liv. Kryptering bygger på ett simpelt koncept, att dölja informationen från all förutom den menade mottagaren. Ett koncept som har funnits med oss under stora delar av människans historia och som än idag är en viktig del av vårt samhälle.<sup>1</sup>

Att dölja information har gått från de enkla metoderna som användes redan för 2000 år sedan för att förmedla hemlig information med hjälp av exempelvis Caesarchiffer till det komplexa och invecklade algoritmer som idag används för att skydda nästan all information som lagras och skickas över internet.<sup>2</sup> Dessa komplexa algoritmer har utvecklats under en tid där datorer står som det dominerande informationshanteringsverktyget och därför är det även en dator som används som huvudverktyg i denna rapport för att undersöka en av det vanligaste krypteringsalgoritmerna i dagens samhälle, The Advanced Encryption Standard (AES).

## 1.1 Syfte

Syftet med undersökning är att undersöka krypteringsalgoritmen AES, för att utveckla en fördjupad förståelse för mer avancerade krypteringsalgoritmer. Samt bygga en uppfattning om hur det på olika sätt går att implementera krypteringsalgoritmer och vilken påverkan detta då får på deras prestanda och säkerhet.

## 1.2 Frågeställningar

- Hur påverkas tiden det tar att kryptera något mellan det olika nyckellängderna 128-bit, 192-bit och 256-bit nyckel?
- Hur förändras krypterings tiden mellan de olika körlägena ECB, CBC & OFB?
- Hur påverkas skiffrertexten av det olika körlägena ECB, CBC & OFB samt vilken betydelse får det ur ett säkerhetsperspektiv?

## 1.3 Avgränsning

Denna undersökning är avgränsad till att endast undersöka AES och dess användning med fokus på hur nyckellängd och körläge påverkar krypteringstiden. Detta samt hur den resulterande skiffer texten påverkas av det olika körlägena ECB, CBC & OFB ur ett säkerhetsperspektiv.

Denna analys av algoritmens säkerhet tar inte hänsyn till faktorer så som möjliga attacker där ibland exempelvis Brute-Force<sup>3</sup> & Side-Channel<sup>4</sup> attacker. Undersökningen är även begränsad till att endast utföras på en mjukvaruimplementering av AES.

---

<sup>1</sup>Dennis Luciano och Gordon Prichett. “Cryptology: From Caesar ciphers to public-key cryptosystems”. I: *The College Mathematics Journal* 18.1 (1987), s. 2–17.

<sup>2</sup>LP87.

<sup>3</sup>Neeraj Kumar. “Investigations in brute force attack on cellular security based on des and aes”. I: *IJCCEM International Journal of Computational Engineering & Management* 14 (2011), s. 50–52.

<sup>4</sup>“Algebraic Side-Channel Attacks on the AES: Why Time also Matters in DPA”. I: *Cryptographic Hardware and Embedded Systems - CHES 2009*. Utg. av Christophe Clavier och Kris Gaj. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, s. 97–111. ISBN: 978-3-642-04138-9.

# 2 Bakgrund

## 2.1 Kryptografi

Ordet kryptografi härstammar från de två grekiska orden kryptos som betyder gömd och grafein som betyder skrift.<sup>5</sup> I sin simplaste form handlar kryptografi alltså om att gömma information. Detta är något som har visat sig på många olika sätt genom historien från något så simpelt som att skriva ett meddelande i text då läskunnigheten var låg till att idag istället använda komplexa algoritmer så som AES & DES.<sup>6</sup> Begreppet kryptografi har dock också fått en utökade betydelse med tiden då det idag även inkluderar olika metoder för att säkerställa autenticiteten av informationen samt identiteten av avsändaren.<sup>7</sup>

## 2.2 Varför behövs kryptering?

I takt med utvecklingen av såväl tekniken som samhälle så visar sig en tydlig trend mot digitalisering av allt från post och meddelanden till betalningar och personuppgifter. Detta har öppnat upp för helt nya problem när det gäller säkerhet och integritet av information som inte tidigare funnits. Utan denna utbredning av digitalisering så hade vår utveckling troligen begränsats men med den nya tekniken kommer även nya problem som måste lösas.<sup>8</sup>

Ett av dessa problem är integritet och säkerhet. Något som tidigare kunde lösas genom att låsa in informationen på en fysisk plats något som inte fungerar när allt ska vara tillgängligt hela tiden oavsett plats. Den digitala världen har alltså gjort det nästan omöjligt att vara helt säker och strävan efter att behålla den enskilda individens integritet är en av de största utmaningarna som vi står inför idag.<sup>9</sup>

### 2.2.1 Kryptografins uppkomst

Kryptografins historia kan man nästan säga börjar vid den tidigaste formen av skrift, vilket grundar sig i de faktum att de flesta inte kunde läsa. Detta är ju såklart något som förändrats på senare tid och i takt med de så har även kryptografin utvecklats. Exempel på utvecklingen går att se så tidigt som 1900 f.Kr då vissa egyptiska skribenter använde sig utav hieroglyfer på ett avvikande sätt, vilket troligen då gjordes i syfte att dölja informationen från dom som inte visste vad det skulle betyda.<sup>10</sup>

Den tidiga kryptografin är även något som kan observeras hos romarna där man använde Cae-sarchiffer och hos grekerna. Där grekernas metod byggde på att man virade en pappersbit runt någon form av ett cylinderformat objekt och sedan skrev meddelandet på pappersbiten. När pappersbiten sedan togs av så är texten oläslig och mottagaren behövde vira upp pappersbiten på ett cylinderformat objekt med samma diameter för att läsa det.<sup>11</sup>

<sup>5</sup>Wikipedia. *Kryptografi*. 2020. URL: <https://sv.wikipedia.org/w/index.php?title=Kryptografi&oldid=48532107> (hämtad 2022-09-07).

<sup>6</sup>Tony M Damico. "A brief history of cryptography". I: *Inquiries Journal* 1.11 (2009).

<sup>7</sup>Nationalencyklopedin. *kryptografi*. 2022. URL: <http://www.ne.se/uppslagsverk/encyklopedi/lng/kryptografi> (hämtad 2022-09-07).

<sup>8</sup>Whitfield Diffie och Susan Landau. *Privacy on the line: The politics of wiretapping and encryption*. The MIT Press, 2010.

<sup>9</sup>DL10.

<sup>10</sup>Dam09.

<sup>11</sup>Dam09.

## 2.2.2 Kryptografins utveckling

Utvecklingen av kryptografin som en vetenskap och teknik såg dock inga större framsteg ända till medeltiden. När utvecklingen ändå började ta fart igen så använde bland annat nästan alla europeiska nationer någon form av kryptografi för att dölja meddelande och hemlig kommunikation. Under den här tiden utvecklades bland annat Polyalphabetic substitutionsskiffer där ett av dom tidigaste skapades av Leon Battista Alberti.<sup>12</sup>

Därefter så fortsattes Polyalphabetic substitutionsskiffer att användas och utvecklas under många år fram till 1900 då bland annat Enigma uppkom. Enigma var ett krypteringsverktyg som bygger på Substitutionsskiffer precis som många skiffer tidigare men som tills skillnad från tidigare använde sig av ett flertal nya metoder för att göra krypteringen säkrare.<sup>13</sup>

Enigma kan man nästan se som ett av de första stege i utvecklingen av den moderna kryptografin som till stora delar bygger på våran teknologiska utveckling. Den nya tekniken öppnade nya portar, vilket bland annat gjorde det möjligt för krypteringen att bli mer komplicerad och säkrare utan att påverka användbarheten. Men utvecklingen visades sig även inom dekrypteringen där ett tydligt exempel är hur en av de första fullt programmerbara datorerna Colossus skapades. Datorn hade i syfte att användes i arbetet med att dekryptera meddelande skickade av tyskarna under andra världskriget och spelade på så sätt en ganska viktig roll i historien.<sup>14</sup>

Senare in på 1900-talet och tidigt 2000-tal så har kryptografin utvecklats ytterligare och idag finns otaliga algoritmer och system som används för att kryptera meddelanden. Däribland bland annat algoritmer som Advanced Encryption Standard (AES) och Data Encryption Standard (DES) men även protokoll som HTTP och SSH.<sup>15</sup>

## 2.3 AES Uppkomst

Startskottet för uppkomsten av AES gavs av National Institute of Standards and Technology (NIST) som 1997 utlyste en utmaning för att skapa en ny standard för kryptering för att ersätta DES som då var den dominerande standarden.<sup>16</sup> Utmaningen utlystes för att DES säkerhet började bli allt mer ifrågasatt i takt med att datorerna blev mer kraftfulla, vilket då blev starten för sökandet efter en ny mer framtidssäker standard.<sup>17</sup>

NIST utlyste sedan 1998 de 15 kandidaterna som valts ut. Därefter så fick den kryptografiska forskargruppen runt om i världen möjligheten att undersöka och testa de olika kandidaterna under processen. Efter ett flertal runder av analysering och testande där antalet kandidater sakta men säkert minskat så valdes till slut 5 kandidater ut som finalister. Dessa var Rijndael, RC6, Serpent, MARS och Twofish. Slutligen en tid senare så valdes Rijndael ut som den nya standarden och en modifierad version av Rijndael blev då sedan den så kallade Advanced Encryption Standard (AES).<sup>18</sup>

---

<sup>12</sup>Dam09.

<sup>13</sup>Dam09.

<sup>14</sup>Wik20.

<sup>15</sup>Wik20.

<sup>16</sup>James Nechvatal m. fl. "Report on the development of the Advanced Encryption Standard (AES)". I: *Journal of research of the National Institute of Standards and Technology* 106.3 (2001), s. 511.

<sup>17</sup>William E Burr. "Selecting the advanced encryption standard". I: *IEEE Security & Privacy* 1.2 (2003), s. 43–52.

<sup>18</sup>Nec+01.

# 3 Teori

## 3.1 Kryptering

Kryptering handlar om att gömma information för att förhindra att andra än den menade mottagaren kan läsa informationen. Detta genomförs i dagens samhälle genom olika typer av krypteringsalgoritmer. Dessa algoritmer kan ses som både komplicerade och förvirrande men bygger på enkla principer. En krypteringsalgoritm tar in en text och en nyckel och ger sedan tillbaka en skiftext, vilket då är en krypterad version av den ursprungliga texten.<sup>19</sup>

För att den menade mottagaren ska kunna läsa skiftexten sedan så behöver hen ha en nyckel samt köra samma krypteringsalgoritm i dekrypterings läge. Bland det vanligaste typerna av krypteringsalgoritmer finns Symetrisk & Asymmetrisk Kryptering, vilket bland annat innehållar algoritmer som AES och RSA.<sup>20</sup>

## 3.2 Blockskiffer

Blockskiffer är en krypteringsalgoritm som verkar på block av data med en fast storlek. Blockskiffer används bland annat som en av grundkomponenterna i kryptografiska protokoll som Transport Layer Security (TLS) och Secure Socket Layer (SSL) som används för att kryptera data som skickas över internet.<sup>21</sup>

En utav det största problemen med Blockskiffer är dock att oavsett hur säkra dom är så passar det bara att använda för kryptering av enskilda block med en nyckel, vilket skiljer sig från något som ett Strömskiffer. På grund av detta har en mängd olika körlägen utvecklats för att på så sätt göra det möjligt att utnyttja blockskiffer för att kryptera större mängder data med en nyckel.<sup>22</sup>

Blockskiffer används däremot inte bara för kryptering utan kan även användas i olika typer av Hashfunktioner och Pseudolumpnummerngeneratorer, eftersom blockskifferets resulterande skiftext ser ut att vara helt slumpmässig trots att den inte egentligen är det.<sup>23</sup>

### 3.2.1 Körlägen

Körlägen inom kryptografin kan man se som algoritmer som appliceras i användningen av blockskiffer eftersom dessa endast är användbara för säker kryptering av ett litet block med bestämd längd. Körlägena gör det då möjligt att istället kunna använda blockskiffer på större datamängder. Detta löser körlägen på olika sätt beroende på hur man vill använda blockskiffer algoritmen samt hur mycket man är villig att kompromissa med säkerheten i förhållande till hastighet.<sup>24</sup>

<sup>19</sup>Wikipedia. *Kryptering*. 2021. URL: <https://sv.wikipedia.org/w/index.php?title=Kryptering&oldid=49187134> (hämtad 2022-09-08).

<sup>20</sup>Wik21.

<sup>21</sup>Wikipedia, the free encyclopedia. *Block cipher*. 2022. URL: [https://en.wikipedia.org/w/index.php?title=Block\\_cipher&oldid=1111913955](https://en.wikipedia.org/w/index.php?title=Block_cipher&oldid=1111913955) (hämtad 2022-10-05).

<sup>22</sup>Wik22c.

<sup>23</sup>Wik22c.

<sup>24</sup>Morris J Dworkin. *Sp 800-38a 2001 edition. recommendation for block cipher modes of operation: Methods and techniques*. 2001.

Ett av sätten som körlägen löser problemet med att applicera blockskiffer algoritmer på större datamängder är att dom använder sig av en så kallad Initialization Vector (IV). Det är en unik sekvens av bytes som används för att säkerställa att samma datamängd aldrig kommer att generera samma krypterade skifffertext.<sup>25</sup>

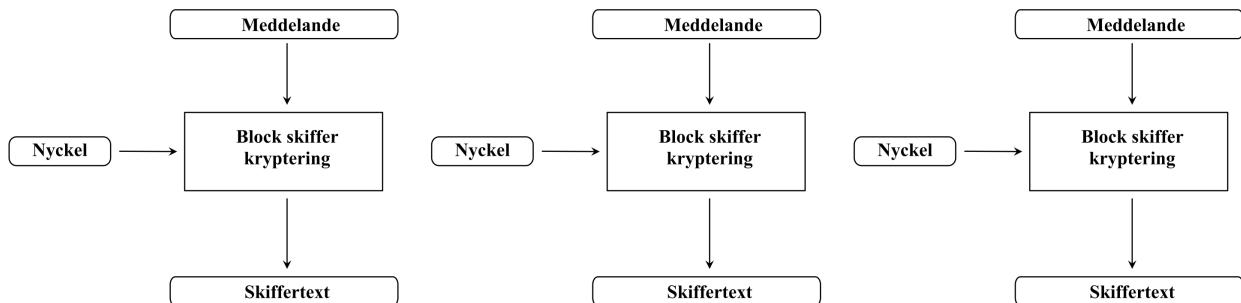
IV kan användas på många sätt, exempelvis kan den introduceras genom en XOR-operation till första blocket. Sedan kedjas resterande block ihop med en XOR-operation med resultatet från den första operationen, vilket precis är vad som görs i CBC. Detta gör att blocken blir beroende av varandra vilket förhindrar att upprepad information som krypteras ger samma resultat även fast man använder samma nyckel.<sup>26</sup>

Exempel på hur ECB, OFB och CBC ser ut implementerade i Python går att se i appendix C.

### 3.2.1.1 ECB

Electronic Code Book läge (ECB) är en av det enklaste blockchiffer körlägena som finns. ECB i sig är ganska lätt att förstå och bygger i huvudsak bara på att man delar upp den data man vill kryptera i delar kallade block och tar sedan varje block för sig och kör genom algoritmen, vilket tydligt visas i figur 3.1 & 3.2.<sup>27</sup>

Figur 3.1 visar hur ECB fungerar vid kryptering. Här visas hur varje block för sig krypteras med hjälp av en blockchiffer algoritm tillsammans med den givna nyckeln.



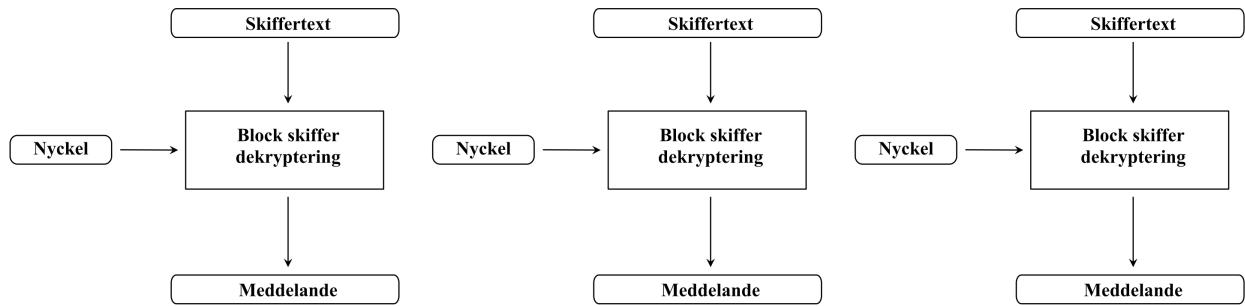
**Figur 3.1:** Electronic Code Book läge kryptering

Figur 3.2 visar istället hur ECB fungerar vid dekryptering, vilken är en i stort sett identisk operation med det enda undantaget att blockchiffret körs i dekrypterings läge istället för krypterings läge.

<sup>25</sup>Dwo01.

<sup>26</sup>Dwo01.

<sup>27</sup>Dwo01.



**Figur 3.2:** Electronic Code Book läge dekryptering

På grund av ECB körlägets enkelhet så finns det dock även ett ganska stort problem med detta körläge. Det handlar om att ECB inte på något sätt förhindrar att två block med samma innehåll som krypteras inte resulterar i ett identiskt krypterat block.<sup>28</sup>

Vad detta innebär är att för större mängder data är att det börjar bildas mönster i skifertexten. Detta är något som väldigt tydligt visar sig ifall man krypterar en bild, vilket går att se när man jämför bilaga A.1 & A.2. Det här faktumet är även varför ECB inte betraktas som ett säkert körläge och näst intill aldrig används i praktiken.<sup>29</sup>

ECB har dock även sina fördelar då det bland annat kan parallelliseras både när det gäller krypteringen och dekrypteringen. Detta samt att ECB även gör det möjligt att slumpmässigt dekryptera enskilda block av en skifertext utan att man behöver dekryptera hela texten.<sup>30</sup>

### 3.2.1.2 CBC

Cipher Block Chaining läget är ett av det mest vanligen använda körlägena för många blockchiffer. Till skillnad från ECB så förhindrar CBC att två block med samma innehåll kan ge samma krypterade block. Detta gör CBC genom att lägga till ett extra steg utöver vad som finns i ECB. Steget är en XOR-operation mellan det krypterade blocket nästkommande block innan de körs genom blockchiffer algoritmen.<sup>31</sup> Matematiskt sett kan detta formuleras så här:

$$\begin{aligned} S_i &= K_n(B_i \oplus S_{i-1}) \\ S_0 &= IV \end{aligned}$$

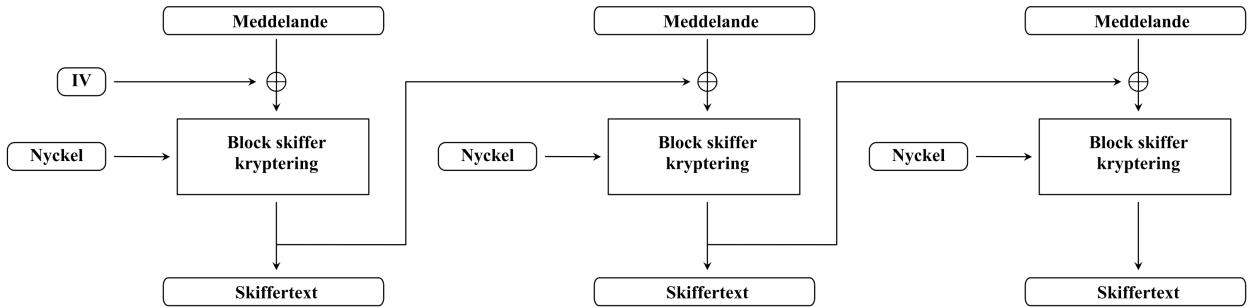
Där  $S_i$  är det krypterade blocket (skifertexten),  $B_i$  är det blocket som ska krypteras,  $K_n$  är blockskifferalgoritmen där  $n$  står för nyckeln och  $S_{i-1}$  är det krypterade blocket före det blocket som ska krypteras. IV är en Initialization Vector (IV) som används vid krypteringen av det första blocket då det inte finns något föregående block att använda.  $i$  står för index där det första blocket har index värdet 1. Hela den här processen kan även ses i figur 3.3.

<sup>28</sup>Dwo01.

<sup>29</sup>Dwo01.

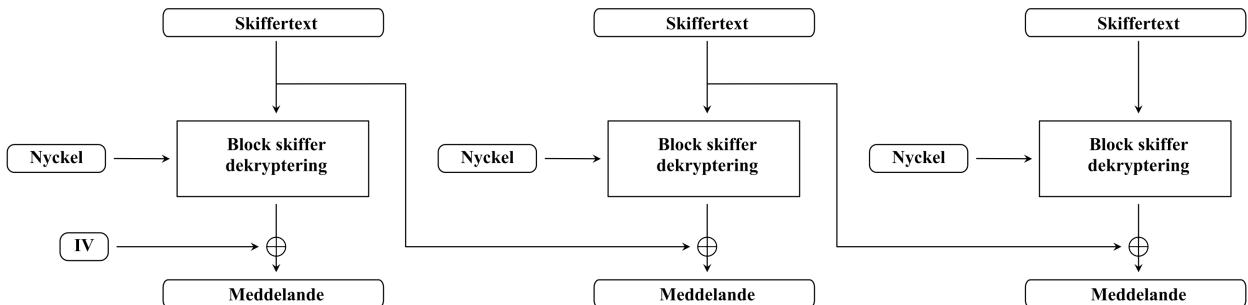
<sup>30</sup>Dwo01.

<sup>31</sup>Dwo01.

**Figur 3.3:** Cipher Block Chaining läge kryptering

När det gäller dekrypteringsprocessen för CBC så bär den precis som för ECB stora likheter med krypteringsprocessen. Det två skillnaderna som finns är att blockskifferet körs i dekrypteringsläge istället för krypteringsläge. Samt att för varje block så genomförs en XOR-operation mellan det dekrypterade blocket och föregående block innan dekrypteringen av blocket.<sup>32</sup> Även detta går att både matematiskt formulera och visuellt visa så här:

$$\begin{aligned} B_i &= K_n(S_i) \oplus S_{i-1} \\ S_0 &= IV \end{aligned}$$

**Figur 3.4:** Cipher Block Chaining läge dekryptering

Fördelarna som kommer från den extra operationen i CBC till skillnad från ECB är då att varje block blir beroende av föregående block. Detta innebär att dom mönster som kunde dyka upp i ECB inte längre kan uppstå, vilket då gör CBC till ett mer säkert körläge än ECB. Dock kräver CBC en ytterligare faktor för att se till så att inte olika meddelanden kan ge samma krypterade block. Därför så krävs en Initialization Vector (IV) som används vid första blocket.<sup>33</sup>

CBC är dock inte perfekt och har i sig också några nackdelar. Där ibland exempelvis det faktum att en inkorrekt IV leder till att det första blocket inte kan dekrypteras korrekt, detta påverkar dock inte det resterande blocken. På grund av det så kan man exempelvis lösa problemet genom att första blocket bara innehåller någon typ av fyllnad, vilket då gör dekrypteringen möjlig utan tillgång till IV.<sup>34</sup>

<sup>32</sup>Dwo01.

<sup>33</sup>Dwo01.

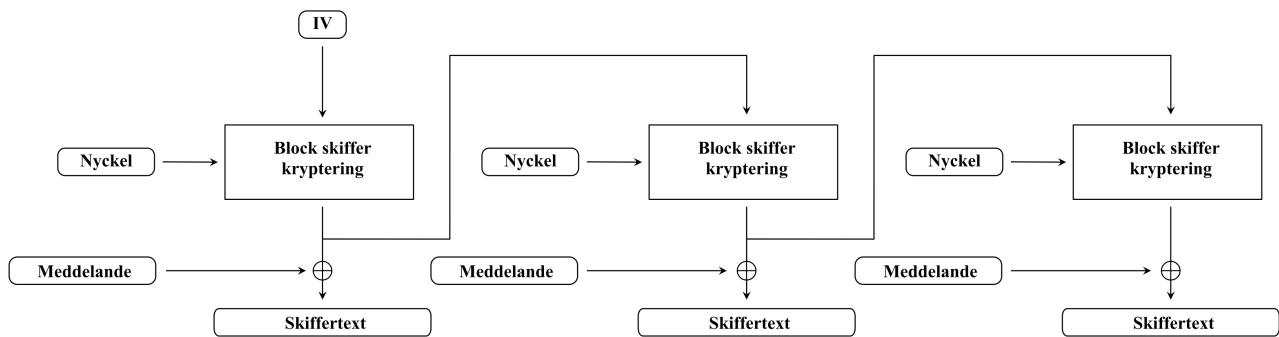
<sup>34</sup>Dwo01.

Utöver detta så begränsas även CBC till att bara gå att parallellisera under dekrypteringen och inte krypteringen, vilket är en konsekvens av att varje block i CBC är beroende av föregående block. CBC behåller dock fortfarande möjligheten som ECB har att slumpmässigt dekryptera enskilda block utan att behöva dekryptera hela skifftexten.<sup>35</sup>

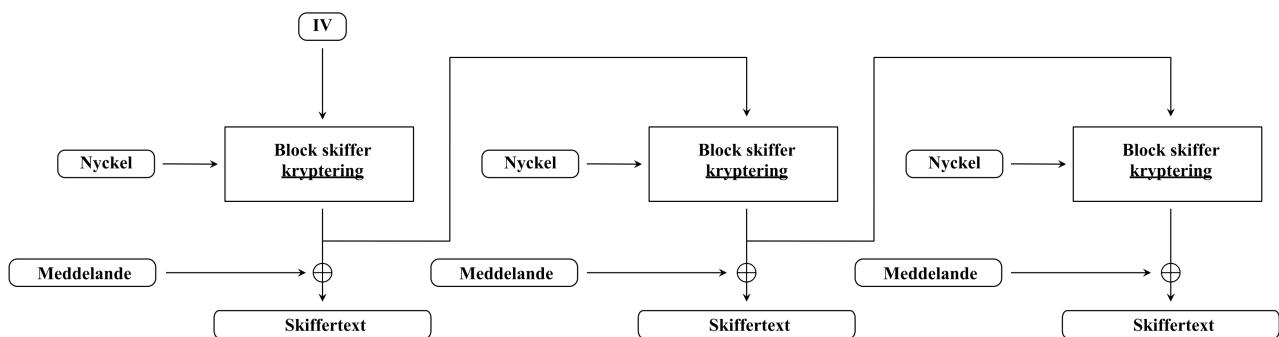
### 3.2.1.3 OFB

Output Feedback läge är ett ytterligare körläge som skiljer sig en del från ECB och CBC som redan presenterats. Den största skillnaden från de andra körlägena är att OFB inte använder blockskifferalgoritmen för att kryptera eller dekryptera blocken. Istället så körs IV genom blockskifferalgoritmen och den resulterande Nyckelström tillförs sedan genom en XOR-operation till blocket som ska krypteras eller dekrypteras.<sup>36</sup>

Tack vare XOR-operationens symmetriska natur så är både krypteringen och dekrypteringen av OFB identisk, vilket även visas i figur 3.5 & 3.6:



**Figur 3.5:** Output Feedback läge kryptering



**Figur 3.6:** Output Feedback läge dekryptering

Utöver detta kan man även matematiskt beskriva OFB, vilket visas i ekvationerna nedan:

<sup>35</sup>Dwo01.

<sup>36</sup>Dwo01.

$$\begin{aligned}
 S_i &= B_i \oplus O_i \\
 B_i &= S_i \oplus O_i \\
 O_i &= K_n(I_i) \\
 I_i &= O_{i-1} \\
 I_0 &= IV
 \end{aligned}$$

Här visas OFB körläget matematiskt där  $S_i$  är det krypterade blocket,  $B_i$  är blocket som ska krypteras och  $I_0$  är IV. Men här finns även  $O_i$  som man kan säga är själva Nyckelströmen som används för att kryptera eller dekryptera blocket.  $O_i$  i sin tur bygger då på att  $O_{i-1}$  körs genom blockskifferalgoritmen igen och sedan används för nästa blocks kryptering.

På grund av att OFB är utformat på det här sättet och att själva blocken som ska krypteras inte används fram till sista steget så är det möjligt att genomföra blockskifferoperationerna i förväg, vilket gör det möjligt att även parallellisera OFB. Dock kan OFB inte parallelliseras ifall man inte gör blockskifferoperationerna i förväg. Utöver detta saknar även OFB möjligheten att slumpmässigt dekryptera enskilda block utan att behöva dekryptera hela skiffertexten.<sup>37</sup>

### 3.3 Symmetrisk & Asymmetrisk Kryptering

Symmetrisk och asymmetrisk kryptering handlar om hur nycklar används i olika krypteringsalgoritmer. För symmetriska krypteringsalgoritmer så betyder detta att samma nyckel är vad som används för både kryptering och dekryptering. Medan asymmetrisk kryptering bygger på att man använder olika nycklar för kryptering och dekrypterings processerna.<sup>38</sup>

De symmetriska krypteringsalgoritmernas huvudsakliga nackdel ligger i det faktum att de krävs en delad känd nyckel mellan båda parter. Detta är något som asymmetriska krypteringsalgoritmer inte behöver, vilket har lett till att man ofta använder asymmetriska krypteringsalgoritmer för att sköta nyckelutbytet för det symmetriska krypteringsalgoritmerna. Anledningen till detta är att det symmetriska krypteringsalgoritmerna ofta är bättre för större datamängder då dom bland annat behöver mycket kortare nyckellängder.<sup>39</sup>

Exempel på symmetriska krypteringsalgoritmer är bland annat AES och DES varav AES kommer förklaras djupare senare i denna rapport.<sup>40</sup> Exempel på asymmetriska krypteringsalgoritmer är bland annat RSA.<sup>41</sup>

### 3.4 AES

Som nämnts tidigare i AES Uppkomst bygger AES-standarden på en variant av Rijndael Blockskiffer algoritmen skapad av Vincent Rijmen och Joan Daemen. AES är en symmetrisk Blockskifferkrypteringsalgoritm som utformades för att ersätta DES som då var den dominerande krypteringsalgoritmen. AES är en av det mest använda krypteringsalgoritmerna idag och

<sup>37</sup>Dwo01.

<sup>38</sup>Wikipedia. *Symmetric-key algorithm*. 2022. URL: [https://en.wikipedia.org/w/index.php?title=Symmetric-key\\_algorithm&oldid=1106743629](https://en.wikipedia.org/w/index.php?title=Symmetric-key_algorithm&oldid=1106743629) (hämtad 2022-09-25).

<sup>39</sup>Wik22b.

<sup>40</sup>Wik22b.

<sup>41</sup>Wikipedia, the free encyclopedia. *RSA*. 2022. URL: <https://sv.wikipedia.org/w/index.php?title=RSA&oldid=50280992> (hämtad 2022-10-04).

används bland annat i säkerhetsprotokoll så som Wi-Fi Protected Access 2 (WPA2)<sup>42</sup>. AES går att dela upp i tre olika varianter som alla är baserade på samma grundläggande struktur. Dessa varianter är AES-128bit, AES-192bit och AES-256bit som huvudsakligen skiljer sig från varandra när det kommer till längden av krypteringsnyckeln samt antalet rundor som genomförs i algoritmens krypterings process.<sup>43</sup>

Även själva algoritmen kan delas upp i ett antal olika delar som alla har olika syften. Dessa delar är AddRoundKey-operationen, SubBytes-operationen, ShiftRows operationen, MixColumns-operationen och Nyckelutökning som alla kommer att förklaras mer detaljerat i det följande sektioner. Nyckelutökning delen kan även delas upp ytterligare i tre olika varianter 128bit, 192bit och 256bit beroende på vilken av det tre AES varianterna som används.<sup>44</sup>

Utöver detta så bygger Nyckelutökning på tre operationer vilka är SubWord, RotWord och Rcon som också kommer att förklaras mer detaljerat i det följande sektioner.<sup>45</sup> För att förstå vissa av dessa delar så kommer det däremot krävas en förklaring av både AES S-Box och Finite Fields först.<sup>46</sup>

### 3.4.1 Finite Fields

Inom bland annat matematiken och datorvetenskapen finns begreppet Finite Fields som även på svenska kallas för ändliga kroppar. Finite Fields är en matematisk koncept som bland annat kan användas ifall man vill kunna utföra aritmetiska operationer som addition, subtraktion, multiplikation och division med ett bestämt antal element. Produkten från varje operation inom ett Finite Field kommer då alltid att vara ett element inom samma Finite Field.<sup>47</sup>

Detta är ett viktigt koncept för AES då AES använder sig av Finite Fields för att utföra vissa av sina operationer. Där anledningen till behovet ligger i att AES jobbar på bytenivå vilket innebär att alla operationer som utförs måste resultera i ett värde som är mellan 0 och 255. AES Finite Field kan då med detta skrivas som  $GF(2^8)$  där  $GF$  står för Galios Field vilket är en annan benämning på Finite Fields.<sup>48</sup> För en mer grundlig matematisk förklaring av Finite Fields så kan man läsa vidare i rapporten *Ändliga kroppar* av Boman, Anna.<sup>49</sup>

### 3.4.2 AES S-Box

AES använder sig av en så kallad S-Box eller med andra ord substitutions box för en del av sina operationer. S-Boxen används huvudsakligen för att byta ut värden i Nyckelutöknings delen av AES algoritmen samt i SubBytes-operationen.<sup>50</sup>

S-boxen består av 256 olika värden som alla är unika och som alla är mellan 0 och 255. Värdena i S-boxen är ordnade i en lista och värdenas position i listan används som index för att kunna

---

<sup>42</sup>Wikipedia, the free encyclopedia. *WPA*. 2021. URL: <https://sv.wikipedia.org/w/index.php?title=WPA&oldid=49187997> (hämtad 2022-10-26).

<sup>43</sup>Wikipedia, the free encyclopedia. *Advanced Encryption Standard*. 2022. URL: [https://en.wikipedia.org/w/index.php?title=Advanced\\_Encryption\\_Standard&oldid=1117488157#See\\_also](https://en.wikipedia.org/w/index.php?title=Advanced_Encryption_Standard&oldid=1117488157#See_also) (hämtad 2022-10-26).

<sup>44</sup>Joan Daemen och Vincent Rijmen. "AES proposal: Rijndael". I: (1999).

<sup>45</sup>Wik22a.

<sup>46</sup>DR99.

<sup>47</sup>Wikipedia, the free encyclopedia. *Finite field*. 2022. URL: [https://en.wikipedia.org/w/index.php?title=Finite\\_field&oldid=1117661014](https://en.wikipedia.org/w/index.php?title=Finite_field&oldid=1117661014) (hämtad 2022-10-27).

<sup>48</sup>Wik22e.

<sup>49</sup>Boman, Anna. *Ändliga kroppar*. 2016.

<sup>50</sup>Wikipedia, the free encyclopedia. *Rijndael S-box*. 2022. URL: [https://en.wikipedia.org/w/index.php?title=Rijndael\\_S-box&oldid=1110299033](https://en.wikipedia.org/w/index.php?title=Rijndael_S-box&oldid=1110299033) (hämtad 2022-10-27).

hitta ett värde i S-boxen. När S-boxen används för att byta ut ett värde så används det värde som ska bytas ut som indexvärde för att kunna hitta rätt värde i S-boxen.<sup>51</sup>

AES S-box genereras med hjälp av AES Finite Field  $GF(2^8)$ . Själva S-boxen är konstant genom hela AES algoritmen och kan därför vara en attackvektor för olika försök till att bryta krypteringen. På grund av detta så finns det även användningar av dynamiska S-boxar som kan genereras utifrån en viss nyckel och skapar då ytterligare ett lager av säkerhet.<sup>52</sup>

AES har två olika S-boxar varav den ena är en invers av den andra. Detta så att operationen går att utföra i båda riktningarna, vilket gör dekryptering av data möjlig.<sup>53</sup> Dessa två S-boxar går att se nedan i figur 3.7 där värdena är representerade i Hexadecimal tal.

63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

**Figur 3.7:** AES S-box (Den vänstra matrisen) & Invers S-box (Den högra matrisen)

### 3.4.3 Struktur

AES är baserad på en design princip kallad SP-network och byggs huvudsakligen upp av fyra olika steg som alla utförs på en matris av 16 Bytes. Dessa fyra steg är SubBytes-operationen, ShiftRows operationen, MixColumns-operationen och AddRoundKey-operationen, vilket visas i figur 3.8. Stegen utförs antingen 10, 12 eller 14 gånger, vilket brukar kallas för rundor.<sup>54</sup>

Beroende på vilken nyckel som används krävs olika antal rundor. En 128bit nyckel kommer att kräva 10 rundor, en 192bit nyckel kommer att kräva 12 rundor och en 256bit nyckel kommer att kräva 14 rundor. Detta brukar man se som det tre olika krypterings lägena AES-128bit AES-192bit och AES-256bit som då är det tre olika nyckelstorlekarna som AES stödjer.<sup>55</sup>

AES-algoritmen har en fast block storlek på 128bit vilket innebär att det alltid kommer att vara 16 Bytes som krypteras åt gången. Absolut först genomförs en initial AddRoundKey-

<sup>51</sup>Wik22l.

<sup>52</sup>Amandeep Singh, Praveen Agarwal och Mehar Chand. "Analysis of Development of Dynamic S-Box Generation". I: *Computer Science and Information Technology* 5 (2017), s. 154–163.

<sup>53</sup>Wik22l.

<sup>54</sup>DR99.

<sup>55</sup>DR99.

operationen där den första rundnyckeln används. Sedan genomförs 9, 11 eller 13 identiska runder. Den sista runden skiljer sig dock från det andra på så sätt att man hoppar över sista MixColumns-operationen då den inte får någon betydelse ur ett kryptografiskt perspektiv för den slutliga skiffrer texten.<sup>56</sup>

En vanlig runda består av följande steg:

1. SubBytes-operationen
2. ShiftRows operationen
3. MixColumns-operationen
4. AddRoundKey-operationen

Den sista runden ser istället ut på följande sätt:

1. SubBytes-operationen
2. ShiftRows operationen
3. AddRoundKey-operationen

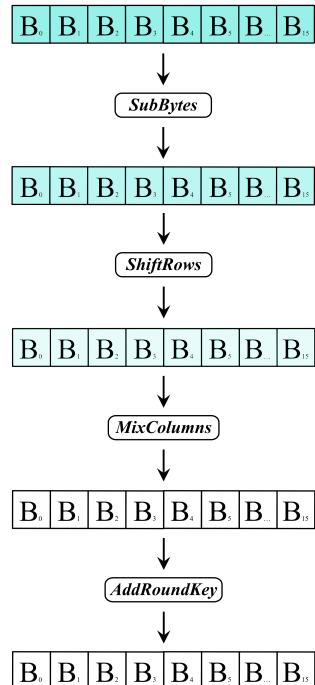
Utifrån detta kan man se att nyckeln används flera gånger under själva krypteringsprocessen av ett block. För att varje runda ska få en unik nyckel så genomförs en Nyckelutöknings operation som genererar en nyckel för varje runda utifrån den ursprungliga nyckeln. Detta för att samma nyckel inte ska användas flera gånger vilket skulle göra det enklare att bryta krypteringen.<sup>57</sup>

Nyckelutökningen är en operationen som genomförs före själva krypteringsstadiet påbörjas och genererar en lista med nycklar för varje runda. Antalet nycklar beror av nyckellängden som även bestämmer antalet runder. För dekryptering så sker liknande runder av operationer som för krypteringen fast i en lite annorlunda ordning samt med den motsatta operationen till ShiftRows operationen & MixColumns-operationen.<sup>58</sup>

En vanlig runda för dekryptering ser ut på följande sätt:

1. AddRoundKey-operationen
2. Inverse MixColumns-operationen
3. Inverse ShiftRows-operationen
4. SubBytes-operationen

Anledningen till att AddRoundKey-operationen och SubBytes-operationen inte kräver någon direkt motsatt funktion har och göra med hur dessa operationer är uppbyggda. För AddRoundKey-operationen så är det på grund av XOR-operationens linjära egenskaper som innebär att den



**Figur 3.8:** Vanlig runda

<sup>56</sup>DR99.

<sup>57</sup>DR99.

<sup>58</sup>DR99.

är sin egen invers. Detta eftersom ifall man tar och genomför XOR-operationen med samma nyckel två gånger så kommer man tillbaka till ursprungsvärdet.

När det gäller SubBytes-operationen så handlar det istället om att det helt enkelt är en substitution utifrån en AES S-Box, vilket då betyder att det som krävs för att genomföra den motsatta operationen blir att använda den motsatta AES S-Boxen.<sup>59</sup>

Både krypterings- och dekrypteringsstrukturen som nu beskrivits kan implementeras i exempelvis Python kod som visas i figur 3.9 & 3.10.

```
# Performs the encryption rounds on the input data matrix
# This function is used for the encryption of data matrixes
# using the expanded keys.
def encryption_rounds(data, round_keys, nr):
    # Inizial add round key
    data = np.bitwise_xor(data, round_keys[0])

    # Rounds 1 to 9 or 1 to 11 or 1 to 13
    # Here each step in one round is performed in a sequence n times
    # where n is the number of rounds minus the last round.
    for i in range(1, (nr - 1)):
        # Sub bytes
        data = sub_bytes(data, subBytesTable)
        # Shift rows
        data = shift_rows(data)
        # Mix columns
        data = mix_columns(data)
        # Add round key
        data = np.bitwise_xor(data, round_keys[i])

    # Final round
    # Identical to the previous rounds, but without mix columns
    data = sub_bytes(data, subBytesTable)
    data = shift_rows(data)
    data = np.bitwise_xor(data, round_keys[nr - 1])

    # Returns the encrypted data
    return data
```

**Figur 3.9:** Krypterings runda funktion

<sup>59</sup>DR99.

```

# Performs the decryption rounds on the input data matrix
# This function is used for the decryption of data matrixes
# using the expanded keys.
def decryption_rounds(data, round_keys, nr):
    # Inizial add round key, inverse shift rows and inverse sub bytes
    data = np.bitwise_xor(data, round_keys[-1])
    data = inv_shift_rows(data)
    data = sub_bytes(data, invSubBytesTable)

    # Rounds 1 to 9 or 1 to 11 or 1 to 13
    # Here each step in one round is performed in a sequence n times
    # where n is the number of rounds minus the last round.
    for i in range(1, (nr - 1)):
        # Add round key
        data = np.bitwise_xor(data, round_keys[-(i+1)])
        # Inverse mix columns
        data = inv_mix_columns(data)
        # Inverse shift rows
        data = inv_shift_rows(data)
        # Inverse sub bytes
        data = sub_bytes(data, invSubBytesTable)

    # Final round
    # Final add round key of final round
    data = np.bitwise_xor(data, round_keys[0])

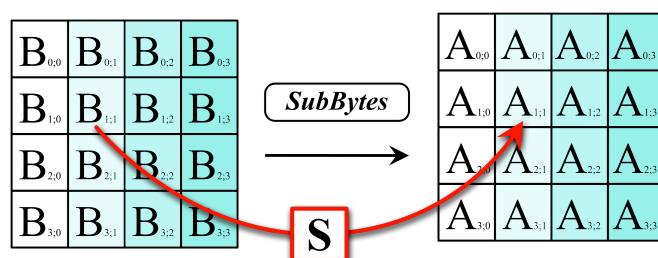
    # Returns the decrypted data
    return data

```

**Figur 3.10:** Dekryptrings runda funktion

### 3.4.3.1 SubBytes-operationen

SubBytes-operationen bygger på att varje Byte byts ut mot en korresponderande Byte från en AES S-Box som visas i figur 3.11. Själva operationen fungerar som det ickelinjära steget i krypteringsprocessen och är därmed en viktig del av AES.<sup>60</sup>

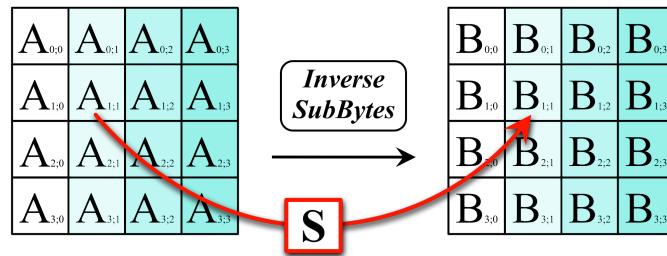


**Figur 3.11:** SubBytes operationen

För att genomföra den motsatta varianten av SubBytes operationen så krävs endast att man använder den motsatta AES S-Boxen, vilket då resulterar i den ursprungliga Byte-matrisen.<sup>61</sup> Vilket visas i figur 3.12.

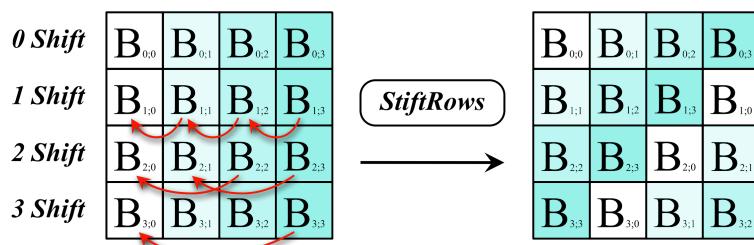
<sup>60</sup>DR99.

<sup>61</sup>DR99.

**Figur 3.12:** Inverse SubBytes operationen

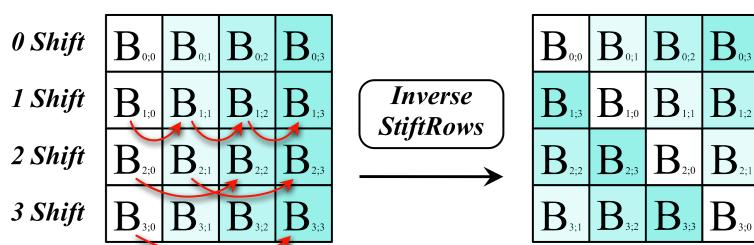
### 3.4.3.2 ShiftRows operationen

ShiftRows operationen är en operation som bygger på att varje rad i Byte matrisen förflyttas i sidled ett visst antal steg. Första raden förflyttas 0 steg, andra raden 1 steg åt vänster, tredje raden 2 steg åt vänster och fjärde raden 3 steg åt vänster. Anledningen till att ShiftRows-steget finns är för att kolumnerna i Byte matrisen krypteras oberoende av varandra.<sup>62</sup>

**Figur 3.13:** ShiftRows operationen

### 3.4.3.3 Inverse ShiftRows-operationen

Denna operationen är den motsatta till ShiftRows-operationen och förflyttar första raden 0 steg, andra raden 1 steg åt höger, tredje raden 2 steg åt höger och fjärde raden 3 steg åt höger.<sup>63</sup> Vilket visas i figur 3.14.

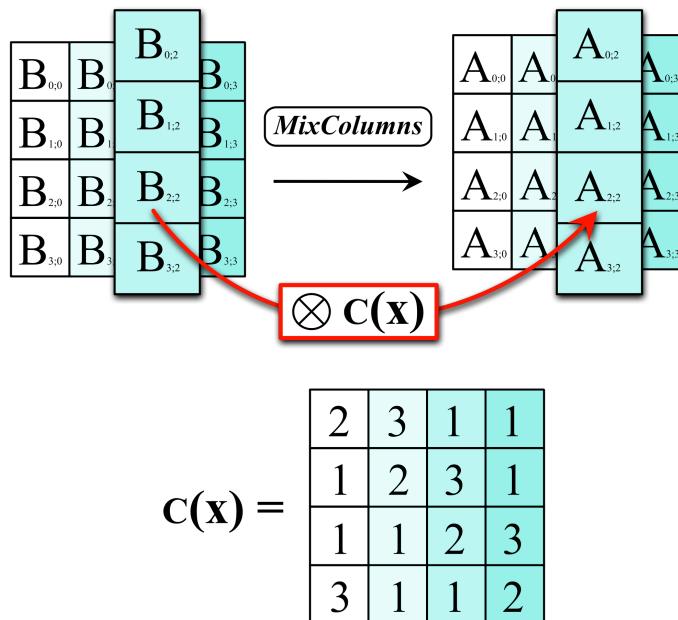
**Figur 3.14:** Inverse ShiftRows operationen

<sup>62</sup>DR99.

<sup>63</sup>DR99.

### 3.4.3.4 MixColumns-operationen

MixColumns-operationen är en operation som primärt agerar på kolumnerna i Byte-matrisen. Operationen bygger på att varje kolumn multipliceras genom en Matrismultiplikation inom  $GF(2^8)$  med en matris som är konstant för alla kolumner. Detta steg gör då att kolumnerna i matrisen omvandlas till nya kolumner, vilket tillsammans med ShiftRows operationen ser till så att inte bara enskilda bytes krypteras utan hela matrisen tillsammans.<sup>64</sup> Själva operationen visas i figur 3.15 där även den konstanta matrisen som multipliceras med varje kolumnerna visas.



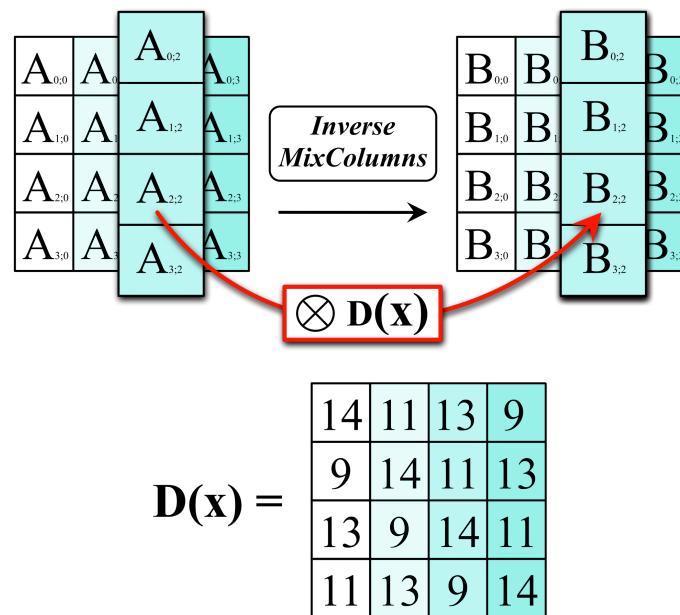
**Figur 3.15:** MixColumns operationen

### 3.4.3.5 Inverse MixColumns-operationen

Inverse MixColumns-operationen är den motsatta operationen till MixColumns-operationen och fungerar på nästan exakt samma sätt. Operationen bygger på att varje kolumn multipliceras genom en matrismultiplikation inom  $GF(2^8)$  med en matris som är konstant för alla kolumner. Den stora skillnaden ligger i att den konstanta matrisen som multipliceras med varje kolumn fungerar som en invers till den tidigare matrisen använd i MixColumns operationen.<sup>65</sup> Själva operationen visas i figur 3.16.

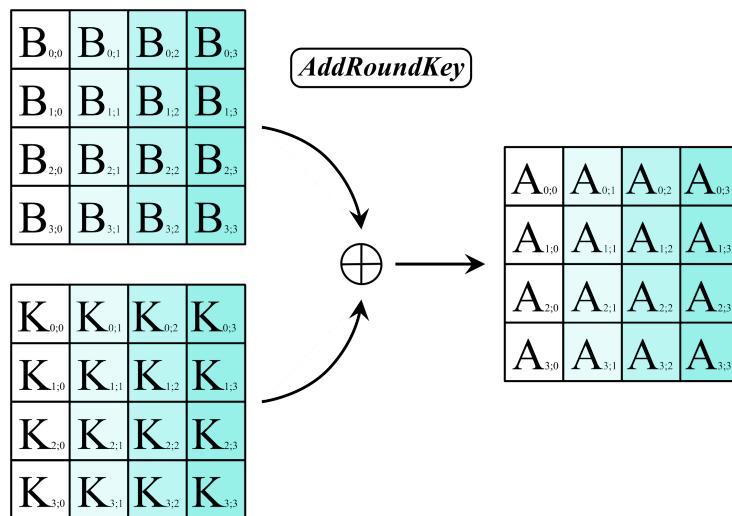
<sup>64</sup>DR99.

<sup>65</sup>DR99.

**Figur 3.16:** Inverse MixColumns operationen

#### 3.4.3.6 AddRoundKey-operationen

AddRoundKey-operationen är en operation som bygger på att varje Byte i Byte-matrisen genomgår en XOR-operation med korresponderande Byte i en nyckelmatris. I det här steget så introduceras nyckeln i krypteringsprocessen vilket är en viktig del av AES som gör det möjligt att kryptera och dekryptera informationen så länge man har rätt nyckel.<sup>66</sup>

**Figur 3.17:** AddRoundKey operationen

---

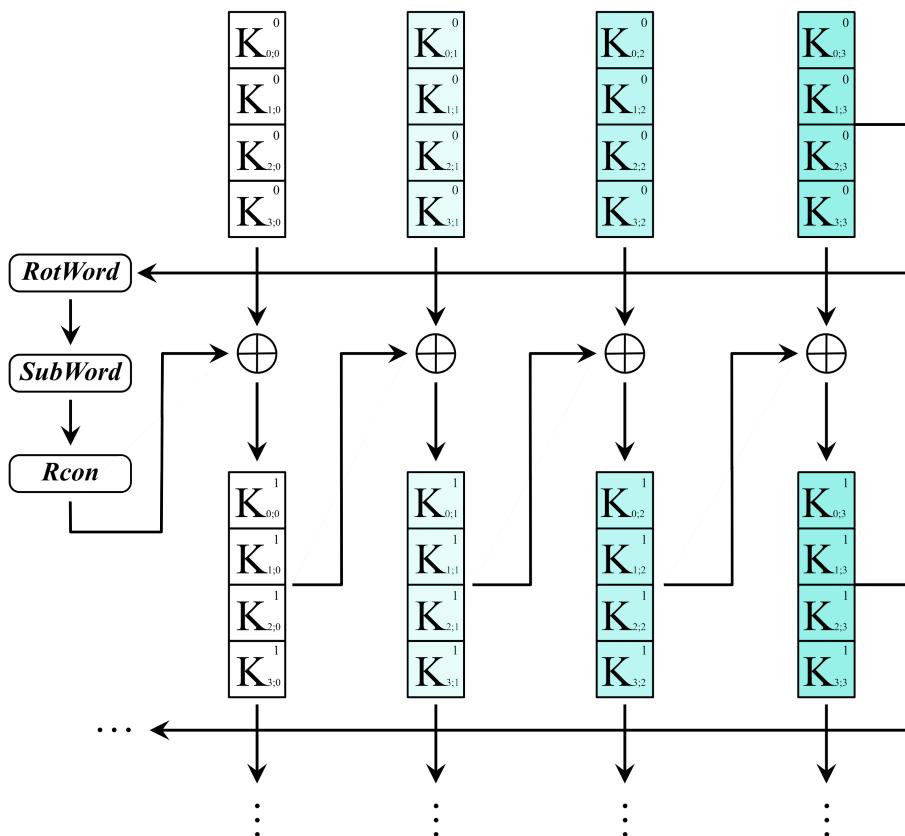
<sup>66</sup>DR99.

### 3.4.4 Nyckelutökning

Nyckelutökningen är den operation som används för att utöka den angivna nyckeln till antingen 11, 13 eller 15 nycklar beroende på längden av den angivna nyckeln. Anledningen till att antalet nycklar inte är den samma som antalet runder är för att det innan första rundan genomförs en inledande AddRoundKey-operationen som använder den första nyckeln. Nyckelutökningen är en viktig del av AES som ser till att det finns en unik nyckel för varje runda i krypteringsprocessen. Själva utökningen av nyckeln genomförs genom att först dela upp den ursprungliga nyckeln i 4 så kallade words.<sup>67</sup>

Varje word består av 4 Byte och är därmed 32 Bits långa. Därefter så genereras varje nytt word utifrån det tidigare worden. Det fyra nästkommande worden genereras genom att ta det sista wordet i den tidigare nyckeln och genom föra en serie av operationer på detta word. Först så genomförs en RotWord-operation och sedan en SubWord-operation samt slutligen en Rcon-operation.<sup>68</sup>

Därefter så genereras det första wordet i nästa nyckel genom en XOR-operation mellan det första wordet i den tidigare nyckeln och det resultat som genererades i föregående steg. Sedan så genereras resterande tre words genom att en XOR-operation genomförs mellan det tidigare wordet och det koresponderande wordet i den tidigare nyckeln. Denna process upprepas tills alla nycklar genererats.<sup>69</sup> Själva processen visas även i figur 3.18.



**Figur 3.18:** Nyckel utöknings processen

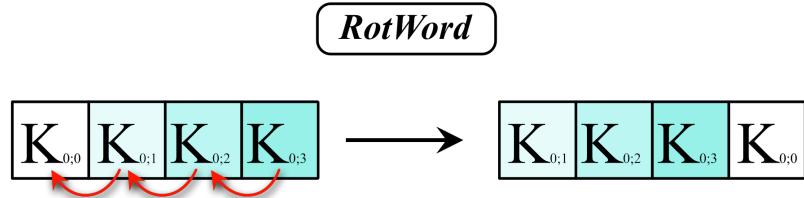
<sup>67</sup>DR99.

<sup>68</sup>DR99.

<sup>69</sup>DR99.

### 3.4.4.1 RotWord

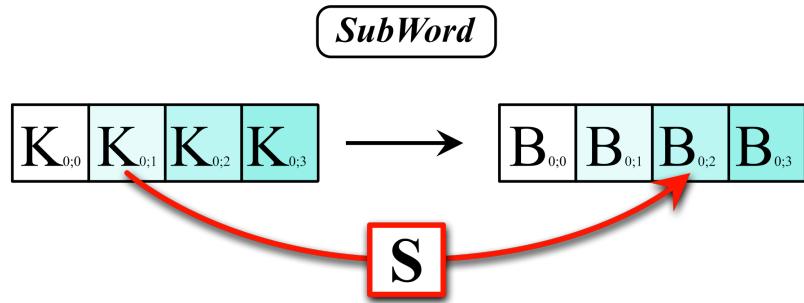
RotWord-operationen är en operation som liknar ShiftRows operationen och fungerar så att varje Byte i wordet skiftas en position till vänster.<sup>70</sup> Detta innebär att den första Byten i wordet hamnar sist och den sista Byten hamnar först, vilket går att se i figur 3.19.



**Figur 3.19:** RotWord operationen

### 3.4.4.2 SubWord

SubWord-operationen är en operation som är nästan identisk med SubBytes-operationen. Själva operationen går ut på att varje Byte i wordet byts ut mot den korresponderande Byten i AES S-Box.<sup>71</sup> Som visas i figur 3.20.



**Figur 3.20:** SubWord operationen

### 3.4.4.3 Rcon

Rcon-operationen bygger på att hela wordet genomgår en XOR-operation med ett speciellt word som är förutbestämt  $[rc_i, 0, 0, 0]$ , vilket går att se i figur 3.21. Det förutbestämda wordet beror vilket nummer nyckeln som genereras är. Exempelvis ifall det är nyckel nummer 2 så kommer det förutbestämda wordet vara  $[1, 0, 0, 0]$ . Det ända som förändras beroende på numret på nyckeln är den första Byten.<sup>72</sup> Värdet på den Byten bestäms genom följande regler:

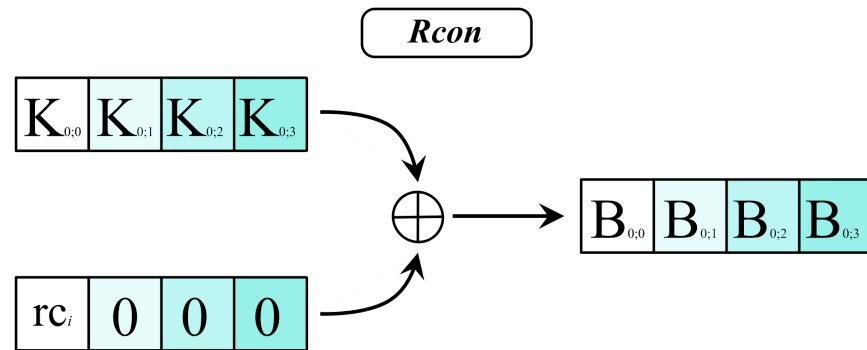
$$rc_i = \begin{cases} 1 & \text{ifall } i = 1 \\ 2 \cdot rc_{i-1} & \text{ifall } i > 1 \text{ och } rc_{i-1} < 80_{16} \\ (2 \cdot rc_{i-1}) \oplus 11B_{16} & \text{ifall } i > 1 \text{ och } rc_{i-1} \geq 80_{16} \end{cases}$$

<sup>70</sup>DR99.

<sup>71</sup>DR99.

<sup>72</sup>DR99.

Där  $\oplus$  är en XOR-operation,  $i$  är vilket nummer på nyckel det är -1 eftersom det första fyra worden är den ursprungliga nyckeln. Utifrån dessa regler för  $rc_i$  så kan man då räkna ut det förutbestämda wordet för varje nyckel som behövs.<sup>73</sup>



**Figur 3.21:** Rcon operationen

---

<sup>73</sup>DR99.

# 4 Metod & Genomförande

Metoden för denna undersökning bygger på en implementering av AES i programmeringsspråket Python. Detta tillsammans med ett antal konstruerade tester, även dom implementerade i Python, är vad som används för själva undersökningen av AES. Koden är skriven med hjälp av programmet VSCode och är byggd huvudsakligen för Python 3.10, men på grund av att Python 3.11 släpptes innan undersökningen genomfördes så är de istället Python 3.11.0 som användes under undersökningsgenomförandet.<sup>74</sup>

## 4.1 Implementering

Implementeringen av AES är uppdelad i ett antal funktioner som till stor del är baserat på hur strukturen och uppdelningen av AES beskrivs i ”AES proposal: Rijndael”<sup>75</sup>. Några av de huvudsakliga funktionerna av algoritmen är SubBytes-operationen, ShiftRows operationen och MixColumns-operationen.

Dessa funktioner används i varje runda och utgör grunden av algoritmen. Utöver detta finns även AddRoundKey-operationen som används för att lägga till nyckeln i varje runda. Sedan finns även en funktion som används för att expandera nyckeln som beskrivs i Nyckelutökning samt individuella funktioner för kryptering och dekryptering till de olika körlägen ECB, CBC och OFB.

Implementeringen använder sig av Python-biblioteket NumPy för att hantera matriserna av Bytes och genomföra matematiska operationer på dessa. Hela implementeringen går att hitta i bilaga C ”AES Implementering i pythonsamt förklaras även i avsnittet AES.”

## 4.2 Testuppsättning

Testuppsättningen är uppdelad i de tre delarna Nyckellängdstest, Körlägestest och Krypterings-test. Två av delarna, Nyckellängdstest och Körlägestest är skrivna som ett Python-script vilket visas i bilaga D. När det gäller Krypteringstest så skiljer sig det från de två andra analyserna eftersom det inte är en automatiserad process utan ett antal praktiska steg som genomfördes för att generera resultatet.

### 4.2.1 Testmiljö

Genomförandet av undersökningen skedde på en Windows 11 dator med Python 3.11.0 installerat. I syfte att minimera variabiliteten mellan de olika omgångarna av tester då allt repeteras ett visst antal gånger så begränsades vissa delar av datorn och anpassningar gjordes i genomförandet. De begränsningar som placerades på datorn bestod av att datorn CPU-klockhastighet begränsades till sin grundhastighet för att ge varje test ett så likvärdigt utgångsläge som möjligt. Utöver detta så genomfördes även varje test separat utan parallellisering för att eliminera risken av att testerna skulle kunna påverka varandra.

---

<sup>74</sup>Python Software Foundation. *Python 3.11.0*. 2022. URL: <https://www.python.org/downloads/release/python-3110/> (hämtad 2022-11-15).

<sup>75</sup>DR99.

### 4.2.2 Nyckellängdstest

Nyckellängdstestet bygger på att varje nyckellängd används för att kryptera en textfil på 1 MB. Detta genomförs 25 gånger för varje nyckellängd och resultatet av varje test sparas sedan i en textfil. Själva testet består av att det för varje runda genereras en textfil på 1 MB fyllt med slumpmässiga Bytes. Denna textfil krypteras sedan med hjälp av AES i körläget ECB samtidigt som tiden som det tar att kryptera filen mätas. Detta upprepas sedan som tidigare nämnts sedan 25 gånger för följande tre nycklarna som går att se nedan.

**Nycklar:**

- 2b7e151628aed2a6abf7158809cf4f3c (128 bit)
- 8e73b0f7da0e6452c810f32b809079e562f8ead2522c6b7b (192 bit)
- 603deb1015ca71be2b73aef0857d77811f352c073b6108d72d9810a30914dff4 (256 bit)

### 4.2.3 Körlägestest

Körlägestestet har en struktur som väldigt mycket liknar det som används i Nyckellängdstest. Det består av att kryptera en textfil på 1 MB fyllt med slumpmässiga bytes med hjälp av AES med den stora skillnaden att istället för att använda olika nycklar så används istället samma nyckel samtidigt som körläget ändras. Krypteringen utförs 25 gånger för varje körläge och resultatet av varje test sparas sedan i en text fil. För att kryptera filen används nyckeln som visas nedan samt för körlägena CBC och OFB så används även IV som visas nedan.

**Nyckel:**

- 2b7e151628aed2a6abf7158809cf4f3c (128 bit)

**IV:**

- 000102030405060708090a0b0c0d0e0f (128 bit)

### 4.2.4 Krypteringstest

Krypteringstestet består utav att en bild krypteras i bildformatet PPM med hjälp av AES i tre olika körlägen. PPM formatet används då de tack vare sin enkla struktur kan genomgå en kryptering och fortfarande representeras som en bild efteråt. Detta gör det då möjligt att visualisera resultatet av krypteringen. För att kryptera bilden används den implementerade AES algoritmen som körs i körlägena ECB, CBC och OFB tillsammans med nyckeln som visas nedan samt för körlägena CBC och OFB så används även IV som visas nedan.

**Nyckel:**

- 2b7e151628aed2a6abf7158809cf4f3c (128 bit)

**IV:**

- 000102030405060708090a0b0c0d0e0f (128 bit)

Före och efter krypteringen modifieras dock PPM filen så att den går att öppna när den väl är krypterad. Detta görs genom att de fyra första raderna i filen tas bort och sätts sedan tillbaka efter krypteringen. Detta gör att filen fortfarande kan öppnas som en bild efter krypteringen.

## 4.3 Genomförande

Först genomfördes Krypteringstestet. För Krypteringstestet valdes först en lämplig bild ut för krypteringen. Sedan konverterades bilden till filformatet PPM med hjälp av programmet GIMP. Därefter utförs följande kommandon i en komandotolk:

```
$ head -n 4 pi.ppm > header.txt  
  
$ tail -n +5 pi.ppm > body.bin  
  
$ python3 encrypt.py 2b7e151628aed2a6abf7158809cf4f3c body.bin ECB  
  
$ cat header.txt body.bin.enc > pi-ecb.ppm
```

Här används filen header.txt som en temporär förvaringsplats för de första fyra raderna i PPM-filen. Detta eftersom de första fyra raderna innehåller informationen som berättar hur filen senare ska tolkas. Därefter används tail-kommandot för att ta bort de första fyra raderna i filen och skriva resten till en ny fil med namnet body.bin. Filen body.bin krypteras sedan med hjälp av Python-filen encrypt.py med nyckeln 2b7e151628aed2a6abf7158809cf4f3c.

Innehållet från den nya filen body.bin.enc sätt sedan ihop med innehållet av header.txt som infogas som de fyra första raderna samt filen döps om till pi-ecb.ppm. Slutligen så konverteras även filen tillbaka till ett mer lättanterligt format med hjälp av GIMP. Processen beskriven ovan upprepades sedan för de två andra körlägen CBC och OFB med den enda skillnaden att de istället för ECB används CBC och OFB som argument till encrypt.py filen (Det tredje terminal kommandot).

Därefter påbörjades Nyckellängdstestet och Körlägestestet, vilket utfördes genom att köra Python filen analyze.py. Filen analyze.py genomförde då krypteringen av en textfil på 1MB fylld med slumpmässigt genererade Bytes med hjälp av AES-implementeringen som går att se i bilaga C. Detta genomfördes i de tre olika körlägena ECB, CBC och OFB samt de tre olika nyckel längderna 128bit, 192bit och 256bit. För varje nyckellängd och körläge så genomfördes krypteringen 25 gånger. Resultatet exporterades därefter till 6 olika text filer, vilkas innehåll sedan sammanställdes i Resultat.

# 5 Resultat

Notera att tiden som visas i tabellerna är i sekunder och är ett medelvärde av 25 enskilda krypteringsomgångar för varje nyckel och varje körläge. För att se tiderna för varje individuell omgång samt bilderna i större storlek se bilaga A, B.1 & B.2.

## 5.1 Nyckellängdstest

Resultat Tid (s)

	ECB - 128bit	ECB - 192bit	ECB - 256bit
Maxvärde	21,71	25,31	29,28
Minvärde	20,84	24,89	29,03
Medelvärde	21,06	25,01	29,18

Procentuell skillnad (medelvärde)

ECB - 128bit & 192bit	18,8%
ECB - 192bit & 256bit	16,6%
ECB - 128bit & 256bit	27,8%

## 5.2 Körlägestest

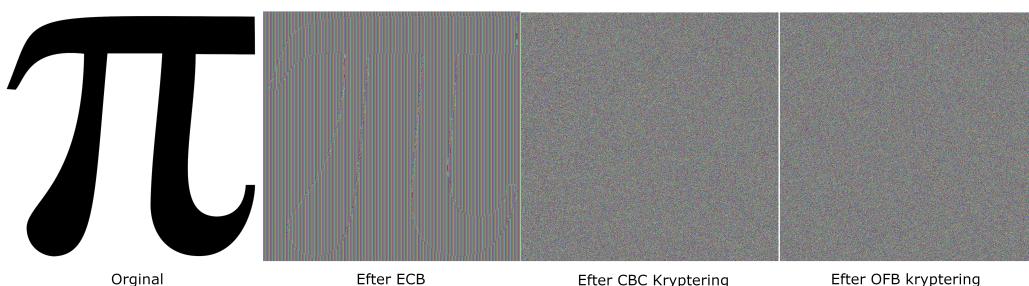
Resultat Tid (s)

	ECB - 128bit	CBC - 128bit	OFB - 128bit
Maxvärde	21,09	20,93	20,88
Minvärde	20,80	20,85	20,77
Medelvärde	20.82	20.89	20.90

Procentuell skillnad (medelvärde)

ECB - 128bit & CBC - 128bit	0,3%
CBC - 128bit & OFB - 128bit	0,1%
ECB - 128bit & OFB - 128bit	0,4%

## 5.3 Krypteringstest



Figur 5.1: AES krypterings test (ECB, CBC, OFB)

# 6 Diskussion

Utifrån resultatet från undersökningen går det att se hur tiden det tar att kryptera en fil på 1MB ökar vid användandet av längre nycklar så som 192-Bit och 256-Bit nycklar i förhållande till en 128-Bit nyckel. Något som bland annat går att se hos procentuella tidsskillnaden mellan det olika nyckel längderna. Där det går att observera en tidsskillnad på 18,8% mellan 128-Bit & 192-Bit nyckeln och 27,8% mellan 128-Bit & 256-Bit nyckeln.

Anledningen till denna tidökning är något som exempelvis kan bero på antalet runder som genomförs för varje 16-Byte block som krypteras, vilket är den huvudsakliga skillnaden mellan det olika nyckel längderna. Eftersom för en 128-Bit nyckel så genomförs 10 runder för varje 16-Byte block som krypteras, medan för en 192-Bit nyckel så genomförs 12 runder och för en 256-Bit nyckel så genomförs 14 runder. Något som då höjer antalet operationer som genomförs för varje 16-Byte block, vilket i sin tur då troligen höjer den totala krypterings tiden.

Resultatet visar även att skillnaden i krypteringstid mellan olika körlägen är väldigt liten. Mellan ECB & CBC var skillnaden 0,3% och mellan ECB & OFB var skillnaden 0,4%. Något som inte riktigt var väntat då det för både CBC och OFB genomförs en ytterligare operation mellan varje 16-Byte block som krypteras för att länka ihop de olika blocken. Men samtidigt så är denna extra operation en XOR-operation som är relativt lätt och snabb för en dator att genomföra. Något som då innebär att trots att det genomförs en extra operation så är den ändå relativt snabb och därfor inte påverkar krypterings tiden särskilt mycket.

I resultatet av körlägestestet kan det dock även observeras viss spridning i tiden som det tar att kryptera en fil mellan det olika omgångarna. Detta bland annat vid jämförelse av max och min tiderna för det olika körlägena. Exempelvis maxvärdet för ECB som är 21,09 s, vilket är högre än maxvärdet för CBC som är 20,93 s och OFB som är 20,88 s. En skillnad som då påvisar att det finns en viss osäkerhet i resultatet, något som även ytterligare bekräftas av min värdena. I och med detta går det inte med säkerhet att konstatera något utifrån resultatet av körläges testet när det gäller vilken betydelse det har för krypterings tiden.

När det gäller hur säkerheten påverkas av det olika körlägena så går det ganska tydligt att se hur ECB är det mest osäkra körläget för större informationsmängder. Detta eftersom det i resultatet kan observes hur det trots kryptering fortfarande går att se spår av den ursprungliga bilden i informationen som krypteras. Något som inte går att göra när bilden istället krypteras med hjälp av CBC eller OFB.

Själva metoden som användes för att genomföra undersökningen bär med sig både för och nackdelar. Bland annat så medför metoden en ökad förståelse för hur Advanced Encryption Standard (AES) fungerar på en låg nivå tack vare det faktum att implementeringen av algoritmen gjordes specifikt för undersökningen. En förståelse som gör det lättare att formulera resonemang och dra slutsatser om hur AES fungerar. Vilket är något som skulle gå förlorat ifall vid användandet av en befintlig implementation av AES. Men att implementera AES på egenhand innebär också att det finns en risk för att implementeringen av algoritmen inte är helt korrekt och är även en tidskrävande process som kräver mycket arbete. Något som kan ses som en nackdel som skulle kunna undvikas ifall en befintlig implementation av AES används.

En fördel med metoden när det gäller just nyckellängdstestet är att säkerheten hos resultatet stärks genom att varje nyckel testas flera gånger, vilket då medför att potentiell slumpmässiga felkällor som kan påverka resultatet minskas. Men samtidigt så innebär detta att tiden som krävs för att genomföra undersökningen ökar. Något som då påverkar hur stora filer som går att testa för att kunna genomföra undersökningen inom en rimlig tidsrymd.

Ytterligare fördelar med metoden är att den till stor del är automatiserad, vilket då minskar den mänskliga faktorns påverkan. Samtidigt som det gör det lättare att repetera undersökningen fler gånger med liten variabilitet, vilket då resulterar i ett mer tillförlitligt resultat. Sedan så är en annan fördel att det lätt går att jämföra och urskilja skillnader i säkerheten för stora datamängder mellan olika körlägen tack vare att en bild användes som testdata. Vilket då ger en tydlig visuell indikation på hur säkerheten påverkas av det olika körlägena.

## 6.1 Felkällor

När det gäller felkällor så finns det bland annat som nämnts tidigare en risk för att implementeringen av AES inte är helt korrekt. Vilket då skulle kunna påverka resultatets tillförlitlighet. Bland annat genom att introducera tidsskillnader mellan exempelvis olika körlägen eller nyckel längder som inte skulle finnas i en korrekt implementering. Detta innebär då att det finns en risk att fel i implementeringen av AES kan påverka resultatet, vilket därmed tillför en viss osäkerhet till resultatet.

En annan felkälla skulle även kunna vara själva resultathanteringen. Något som för denna undersökning gjordes manuellt efter att undersökningen var genomförd. Detta innebär då att det finns en möjlighet för fel som beror på den mänsklig faktorn som exempelvis felaktig avläsning eller felaktiga beräkningar vid sammanställning av resultatet. Något som då ytterligare påverkar resultatets tillförlitlighet negativt.

Sedan skulle en annan felkälla även kunna vara en variation i CPU-klockhastighet mellan omgångarna, vilket då resulterar i att olika omgångar av undersökningen hinner olika många instruktioner per sekund. En faktor som då innebär att det kan ta olika lång tid för samma operationer att utföras mellan omgångarna i undersökningen. Detta leder då till en viss variation i resultatet, vilket då även till för en viss osäkerhet.

Slutligen skulle även ytterligare en felkälla kunna vara Nyckelutökningen som användes för att generera nycklarna för varje runda. Detta eftersom detta steg behöver genomföra fler operationer för längre nyckellängder. Något som då innebär att det tar längre tid för större nycklar att utökas i förhållande till kortare nycklar. Detta är då en systematisk felkälla som innebär att det alltid kommer finnas ett visst tidstillägg för större nycklar i förhållande till kortare nycklar. Ett faktum som behöver tas i beaktande vid jämförelser mellan olika nyckel längder.

## 6.2 Förbättringar

När det gäller möjliga förbättringar av undersökningen så skulle en förbättring kunna vara att öka antalet gånger som varje nyckel och körläge testas. Detta skulle då minska risken för att resultatet blir påverkat av slumpmässiga felkällor så som exempelvis variationer i klockhastigheten. Något som då skulle göra resultatet mer tillförlitligt.

En annan förbättring skulle även kunna vara att genomföra Nyckelutökningen innan själva tids tagningen av krypteringen. Något som då skulle kunna eliminera den systematiska felkälla som nämnts tidigare när det gäller Nyckelutökningen. Detta innebär då att jämförelser av tiden det tar att kryptera något mellan olika nyckellängder skulle spegla den faktiska skillnaden bättre och därmed ge ett mer tillförlitligt resultat.

Det är dock värt att notera att denna skillnad som kan uppstå mellan olika nyckel längder för Nyckelutökningen inte är särskilt stor i förhållande till resten av krypteringsprocessen i

takt med att data mängden ökar. Detta innebär då att felet som uppstår från att genomföra Nyckelutökningen medan tidtagningen pågår även skulle kunna minimeras genom att öka datamängden. Något som då skulle göra att skillnaden blir försumbar i förhållande till resten av krypteringsprocessen.

Ytterligare en förbättring kan vara att på något sätt begränsa CPU-klockhastighet till en viss fast klockhastighet, vilket då skulle ge varje omgång av undersökningen samma förutsättningar att hinna med exakt samma antal operationer per sekund. Något som då skulle innebära ett mer tillförlitligt resultat. Utöver detta skulle även den mänskliga faktorns påverkan på resultatet kunna minskas genom att automatisera resultathanteringen och beräkningarna av exempelvis medelvärde. Något som då även det skulle göra resultatet mer tillförlitligt.

En annan förbättring skulle även kunna vara att kontrollera implementeringen av AES ytterligare exempelvis genom att låta någon utomstående individ insatt i ämnet och med en bra förståelse av programmeringsspråket Python granska koden. Något som då skulle kunna minimerar risken för felaktigheter i implementeringen, vilket i sin tur då skulle leda till att resultatet blir mer tillförlitligt.

## 6.3 Slutsats

Ska nu frågeställningen “Hur påverkas tiden det tar att kryptera något mellan det olika nyckel längderna 128-bit, 192-bit och 256-bit nyckel?” besvaras så visar resultatet från undersökningen tydligt hur tiden det tar att kryptera något ökar ganska mycket i takt med att nyckeln blir längre. En tids ökning på 18,8% mellan 128-bit och 192-bit nyckel och en tids ökning på 27,8% mellan 192-bit och 256-bit nyckel. Ett resultat som trots vissa felkällor och osäkerheter ändå är tillräckligt tillförlitligt för att kunna dra slutsatsen att det tar längre tid.

Sedan när frågeställningen “Hur förändras krypterings tiden mellan de olika körlägena ECB, CBC & OFB?” ska besvaras så är resultatet inte riktigt lika tydligt som när det gäller nyckel längderna. Resultatet visar på en tidsökning på 0,3% mellan ECB och CBC och en tidsökning på 0,1% mellan CBC och OFB. Att dra några konkreta slutsatser från detta resultat är dock inte möjligt utifrån denna undersökning då det är svårt att säga om skillnaden mellan ECB och CBC samt CBC och OFB är på grund av en faktisk tidsskillnad eller om det är ett resultat av felkällor och osäkerheter. Ett faktum som ganska tydligt framgår i hur skillnaden mellan de högsta uppmätta värdena och det lägst uppmätta värdena för varje körläge. Där det går att se hur exempelvis det högsta värdet för ECB är högre än både det högsta värdet för CBC och det högsta värdet för OFB, samtidigt som det lägsta värdet för OFB är lägre än både det lägsta värdet för ECB och det lägsta värdet för CBC.

Slutligen kan frågeställningen “Hur påverkas skiffertexten av det olika körlägena ECB, CBC & OFB samt vilken betydelse får det ur ett säkerhetsperspektiv?” besvaras med hjälp av resultatet från krypteringstestet där det tydligt går att se hur skiffertexten förändras beroende på vilket körläge som används. Något som visar sig hos skiffertexten från ECB krypteringen där det tydligt går att se spår av den ursprungliga bilden samtidigt som detta inte går att se i skiffertexten från CBC och OFB. Utifrån detta går det då att dra slutsatsen att körläget ECB är sämre ur ett säkerhetsperspektiv för större datamängder än CBC och OFB eftersom det trots kryptering fortfarande går att se spår av den ursprungliga informationen i skiffertexten.

Sammanfattningsvis går det utifrån denna undersökning då att dra slutsatsen att det tar längre tid att kryptera information vid användande av en längre så som en 256-bit nyckel jämfört med en kortare så som 128-bit nyckel. Samtidigt som det möjliga skulle kunna finnas en liten

tidsskillnad mellan ECB, CBC och OFB. Samt att ECB är sämre ur ett säkerhetsperspektiv för att kryptera större informationsmängder än CBC och OFB.

# Källförteckning

- [Abd19] Christian Abdelmassih. *Matriser*. 2019. URL: <https://ludu.co/course/linjär-algebra/matriser/> (hämtad 2023-01-28).
- [BLFF96] Tim Berners-Lee, Roy Fielding och Henrik Frystyk. *Hypertext transfer protocol—HTTP/1.0*. Tekn. rapport. 1996.
- [Bar+01] Daniel J Barrett m. fl. *SSH, the Secure Shell: the definitive guide*. Ö'Reilly Media, Inc.", 2001.
- [Bom16] Boman, Anna. *Ändliga kroppar*. 2016.
- [Bur03] William E Burr. "Selecting the advanced encryption standard". I: *IEEE Security & Privacy* 1.2 (2003), s. 43–52.
- [CG09] "Algebraic Side-Channel Attacks on the AES: Why Time also Matters in DPA". I: *Cryptographic Hardware and Embedded Systems - CHES 2009*. Utg. av Christophe Clavier och Kris Gaj. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, s. 97–111. ISBN: 978-3-642-04138-9.
- [DL10] Whitfield Diffie och Susan Landau. *Privacy on the line: The politics of wiretapping and encryption*. The MIT Press, 2010.
- [DR99] Joan Daemen och Vincent Rijmen. "AES proposal: Rijndael". I: (1999).
- [Dam09] Tony M Damico. "A brief history of cryptography". I: *Inquiries Journal* 1.11 (2009).
- [Dwo01] Morris J Dworkin. *Sp 800-38a 2001 edition. recommendation for block cipher modes of operation: Methods and techniques*. 2001.
- [Fil22] FileInfo.com. *.PPM File Extension*. 2022. URL: <https://fileinfo.com/extension/ppm> (hämtad 2022-11-30).
- [Gim22] Gimp.org. *About GIMP*. 2022. URL: <https://www.gimp.org/about/> (hämtad 2022-11-30).
- [Kum11] Neeraj Kumar. "Investigations in brute force attack on cellular security based on des and aes". I: *IJCEM International Journal of Computational Engineering & Management* 14 (2011), s. 50–52.
- [LEW12] FEATURE MICHAEL LEWIN. "All about XOR". I: *For details of ACCU, our publications and activities, visit the ACCU website: www.accu.org* (2012), s. 14.
- [LP87] Dennis Luciano och Gordon Prichett. "Cryptology: From Caesar ciphers to public-key cryptosystems". I: *The College Mathematics Journal* 18.1 (1987), s. 2–17.
- [Nat22a] Nationalencyklopedin. *hexadecimal*. 2022. URL: <https://www.ne.se/uppslagsverk/encyklopedi/lng/hexadecimal> (hämtad 2022-11-29).
- [Nat22b] Nationalencyklopedin. *kryptografi*. 2022. URL: <http://www.ne.se/uppslagsverk/encyklopedi/lng/kryptografi> (hämtad 2022-09-07).
- [Nec+01] James Nechvatal m. fl. "Report on the development of the Advanced Encryption Standard (AES)". I: *Journal of research of the National Institute of Standards and Technology* 106.3 (2001), s. 511.
- [Pyt22a] Python Software Foundation. *Python 3.11.0*. 2022. URL: <https://www.python.org/downloads/release/python-3110/> (hämtad 2022-11-15).
- [Pyt22b] Python Software Foundation. *What is Python?* 2022. URL: <https://docs.python.org/3/faq/general.html#what-is-python> (hämtad 2022-09-01).
- [SAC17] Amandeep Singh, Praveen Agarwal och Mehar Chand. "Analysis of Development of Dynamic S-Box Generation". I: *Computer Science and Information Technology* 5 (2017), s. 154–163.
- [Wik20] Wikipedia, the free encyclopedia. *Byte*. 2020. URL: <https://sv.wikipedia.org/w/index.php?title=Byte&oldid=48406212> (hämtad 2022-10-05).

- [Wik20] Wikipedia. *Kryptografi*. 2020. URL: <https://sv.wikipedia.org/w/index.php?title=Kryptografi&oldid=48532107> (hämtad 2022-09-07).
- [Wik21a] Wikipedia, the free encyclopedia. *Binära talsystemet*. 2021. URL: [https://sv.wikipedia.org/w/index.php?title=Binra\\_talsystemet&oldid=49765783](https://sv.wikipedia.org/w/index.php?title=Binra_talsystemet&oldid=49765783) (hämtad 2022-10-17).
- [Wik21b] Wikipedia, the free encyclopedia. *Caesarchiffer*. 2021. URL: <https://sv.wikipedia.org/w/index.php?title=Caesarchiffer&oldid=48885737> (hämtad 2022-09-23).
- [Wik21c] Wikipedia, the free encyclopedia. *Keystream*. 2021. URL: <https://en.wikipedia.org/w/index.php?title=Keystream&oldid=1039345792> (hämtad 2022-10-04).
- [Wik21d] Wikipedia, the free encyclopedia. *Visual Studio Code*. 2021. URL: [https://sv.wikipedia.org/w/index.php?title=Visual\\_Studio\\_Code&oldid=48905230](https://sv.wikipedia.org/w/index.php?title=Visual_Studio_Code&oldid=48905230) (hämtad 2022-10-04).
- [Wik21e] Wikipedia, the free encyclopedia. *WPA*. 2021. URL: <https://sv.wikipedia.org/w/index.php?title=WPA&oldid=49187997> (hämtad 2022-10-26).
- [Wik21] Wikipedia. *Kryptering*. 2021. URL: <https://sv.wikipedia.org/w/index.php?title=Kryptering&oldid=49187134> (hämtad 2022-09-08).
- [Wik22a] Wikipedia, the free encyclopedia. *Advanced Encryption Standard*. 2022. URL: [https://en.wikipedia.org/w/index.php?title=Advanced\\_Encryption\\_Standard&oldid=1117488157#See\\_also](https://en.wikipedia.org/w/index.php?title=Advanced_Encryption_Standard&oldid=1117488157#See_also) (hämtad 2022-10-26).
- [Wik22a] Wikipedia. *Operativsystem*. 2022. URL: <https://sv.wikipedia.org/w/index.php?title=Operativsystem&oldid=51604064> (hämtad 2023-01-21).
- [Wik22b] Wikipedia, the free encyclopedia. *Bit*. 2022. URL: <https://sv.wikipedia.org/w/index.php?title=Bit&oldid=51073011> (hämtad 2022-10-05).
- [Wik22b] Wikipedia. *Symmetric-key algorithm*. 2022. URL: [https://en.wikipedia.org/w/index.php?title=Symmetric-key\\_algorithm&oldid=1106743629](https://en.wikipedia.org/w/index.php?title=Symmetric-key_algorithm&oldid=1106743629) (hämtad 2022-09-25).
- [Wik22c] Wikipedia, the free encyclopedia. *Block cipher*. 2022. URL: [https://en.wikipedia.org/w/index.php?title=Block\\_cipher&oldid=1111913955](https://en.wikipedia.org/w/index.php?title=Block_cipher&oldid=1111913955) (hämtad 2022-10-05).
- [Wik22d] Wikipedia, the free encyclopedia. *Enigma machine*. 2022. URL: [https://en.wikipedia.org/w/index.php?title=Enigma\\_machine&oldid=1109006355](https://en.wikipedia.org/w/index.php?title=Enigma_machine&oldid=1109006355) (hämtad 2022-10-17).
- [Wik22e] Wikipedia, the free encyclopedia. *Finite field*. 2022. URL: [https://en.wikipedia.org/w/index.php?title=Finite\\_field&oldid=1117661014](https://en.wikipedia.org/w/index.php?title=Finite_field&oldid=1117661014) (hämtad 2022-10-27).
- [Wik22f] Wikipedia, the free encyclopedia. *Frequency analysis*. 2022. URL: [https://en.wikipedia.org/w/index.php?title=Frequency\\_analysis&oldid=1113626431](https://en.wikipedia.org/w/index.php?title=Frequency_analysis&oldid=1113626431) (hämtad 2022-10-10).
- [Wik22g] Wikipedia, the free encyclopedia. *Hashfunktion*. 2022. URL: <https://sv.wikipedia.org/w/index.php?title=Hashfunktion&oldid=50669121> (hämtad 2022-10-07).
- [Wik22h] Wikipedia, the free encyclopedia. *Logical connective*. 2022. URL: [https://en.wikipedia.org/w/index.php?title=Logical\\_connective&oldid=1113164184](https://en.wikipedia.org/w/index.php?title=Logical_connective&oldid=1113164184) (hämtad 2022-10-12).
- [Wik22i] Wikipedia, the free encyclopedia. *Polyalphabetic cipher*. 2022. URL: [https://en.wikipedia.org/w/index.php?title=Polyalphabetic\\_cipher&oldid=1113775685](https://en.wikipedia.org/w/index.php?title=Polyalphabetic_cipher&oldid=1113775685) (hämtad 2022-10-10).
- [Wik22j] Wikipedia, the free encyclopedia. *Pseudorandomness*. 2022. URL: <https://en.wikipedia.org/w/index.php?title=Pseudorandomness&oldid=1112429322> (hämtad 2022-10-04).
- [Wik22k] Wikipedia, the free encyclopedia. *RSA*. 2022. URL: <https://sv.wikipedia.org/w/index.php?title=RSA&oldid=50280992> (hämtad 2022-10-04).

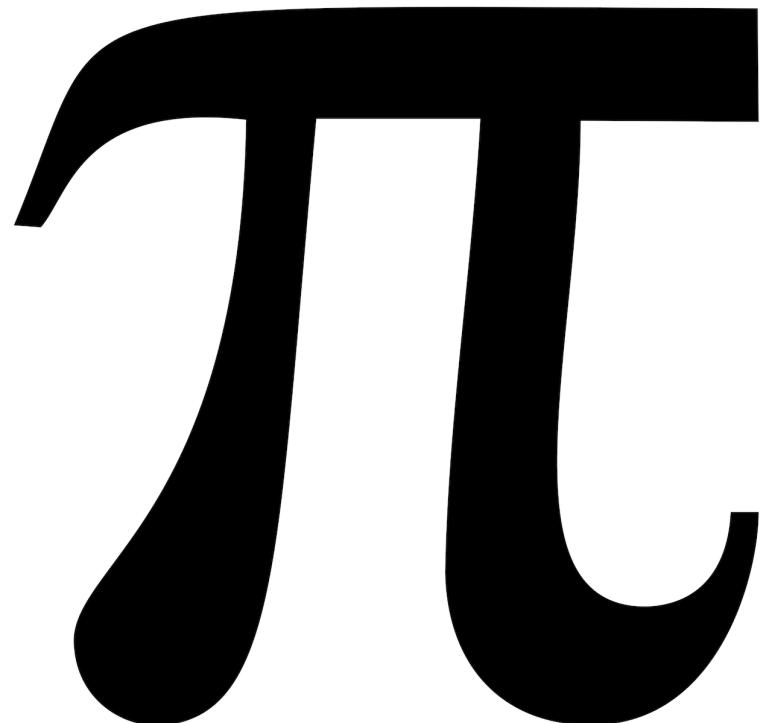
## KÄLLFÖRTECKNING

---

- [Wik22l] Wikipedia, the free encyclopedia. *Rijndael S-box*. 2022. URL: [https://en.wikipedia.org/w/index.php?title=Rijndael\\_S-box&oldid=1110299033](https://en.wikipedia.org/w/index.php?title=Rijndael_S-box&oldid=1110299033) (hämtad 2022-10-27).
- [Wik22m] Wikipedia, the free encyclopedia. *Stream cipher*. 2022. URL: [https://en.wikipedia.org/w/index.php?title=Stream\\_cipher&oldid=1098861130](https://en.wikipedia.org/w/index.php?title=Stream_cipher&oldid=1098861130) (hämtad 2022-10-05).
- [Wik22n] Wikipedia, the free encyclopedia. *Substitution cipher*. 2022. URL: [https://en.wikipedia.org/w/index.php?title=Substitution\\_cipher&oldid=1111925704](https://en.wikipedia.org/w/index.php?title=Substitution_cipher&oldid=1111925704) (hämtad 2022-10-10).
- [Wik22o] Wikipedia, the free encyclopedia. *Substitution-permutation network*. 2022. URL: [https://en.wikipedia.org/w/index.php?title=Substitution-permutation\\_network&oldid=1098155951](https://en.wikipedia.org/w/index.php?title=Substitution-permutation_network&oldid=1098155951) (hämtad 2022-11-10).
- [Wik23a] Wikipedia. *Clock rate*. 2023. URL: [https://en.wikipedia.org/w/index.php?title=Clock\\_rate&oldid=1134817080](https://en.wikipedia.org/w/index.php?title=Clock_rate&oldid=1134817080) (hämtad 2023-01-21).
- [Wik23b] Wikipedia. *Windows 11*. 2023. URL: [https://en.wikipedia.org/w/index.php?title=Windows\\_11&oldid=1134890726](https://en.wikipedia.org/w/index.php?title=Windows_11&oldid=1134890726) (hämtad 2023-01-21).

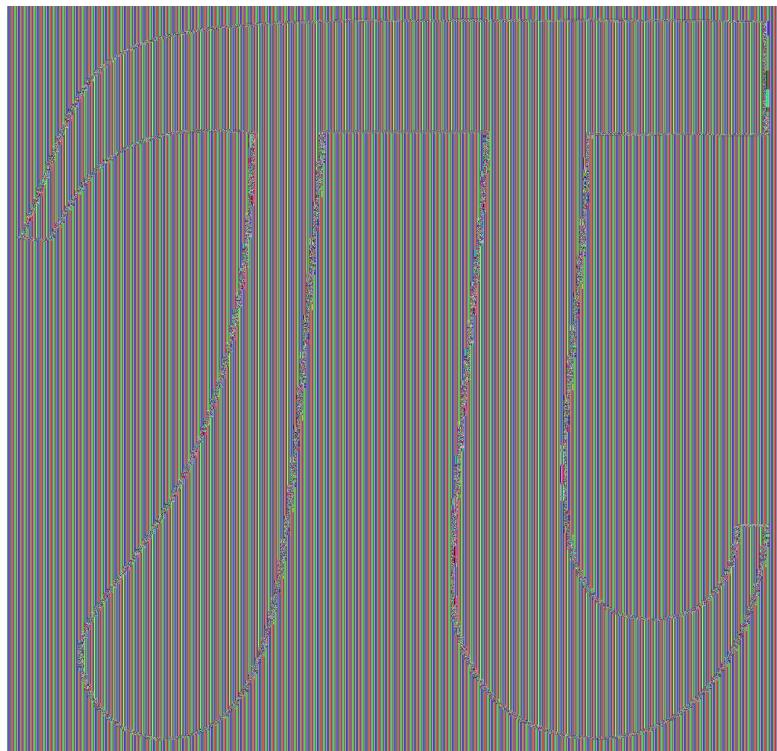
# A AES körlägestest resultat

## A.1 Före testet



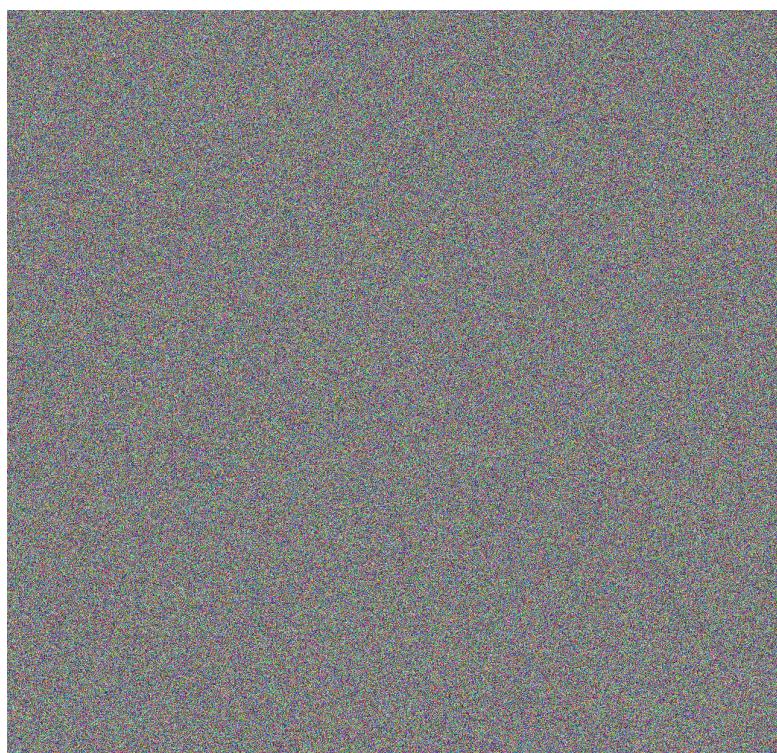
**Figur A.1:** Orginal bild

## A.2 Efter ECB kryptering



**Figur A.2:** Efter ECB Kryptering

## A.3 Efter CBC kryptering



**Figur A.3:** Efter CBC Kryptering

## A.4 Efter CFB kryptering



**Figur A.4:** Efter OFB Kryptering

# B Fullständig data från testerna

## B.1 Data från nyckellängdstest

Resultat Tid (s)		
ECB - 128bit	ECB - 192bit	ECB - 256bit
21.114332500001183	25.019810000005236	29.25573820000136
21.2294222000055	25.03902250000101	29.199436000002606
21.400101300001552	24.921688300004462	29.1446350000042
21.394803399998636	25.010479399999895	29.276371100000688
21.410092899997835	25.023025200003758	29.208361999997578
21.255479499996	24.94711079999979	29.12308629999461
21.709606899996288	24.932161500000802	29.057567500000005
20.930029600000125	24.93615460000001	29.23549970000022
20.91164679999929	25.07268920000206	29.163480200004415
21.024735999999393	25.05104990000109	29.13999270000204
20.974247600002855	24.92792770000233	29.03159289999894
20.906690399999206	25.040346800000407	29.238432199999806
20.949946799999452	25.02671339999506	29.226332500002172
20.96879309999349	24.999294099994586	29.13317239999742
20.955687699999544	24.973698000001605	29.119369399995776
20.907472200000484	24.992430599995714	29.273320899999817
20.99763510000048	25.22444600000017	29.147454799996922
20.964201399998274	25.310352100001182	29.133893199999875
20.970198600000003	25.03863189999538	29.14858010000171
21.033452000003308	24.887886799995613	29.20511969999643
20.835713899999973	25.002411200002825	29.214967400002934
20.933525599997665	25.029954099998577	29.155635700000857
20.899700900001335	24.92795739999565	29.136839000006148
20.861958300003607	25.015061800004332	29.250083099999756
20.898621000000276	25.012236000002304	29.22063089999574

## B.2 Data från körlägestest

Resultat Tid (s)		
ECB	CBC	OFB
20.855055499996524	20.863876399998844	20.801168300000427
20.88846609999746	20.907602499995846	20.78754040000058
20.879108299996005	20.86540999999852	20.823867599996447
20.801008999995247	20.896374500000093	20.87718119999772
20.85887470000307	20.868871400001808	20.770114799997828
20.87116899999819	20.882914800000435	20.767576799997187
20.854077200005122	20.89511720000155	20.85902029999852
20.85488130000158	20.85610540000198	20.797623000005842
20.826587200004724	20.865844100000686	20.796596600004705
20.909639099998458	20.85497480000049	20.797125700002653
20.870294899999863	20.906652400000894	20.835499799999525
20.818692400003783	20.92016809999768	20.865253799995116
20.82849349999742	20.908798900003603	20.800847200000135
20.86879109999427	20.850567699999374	20.795617399999173
20.91046159999678	20.883563499999582	20.834485100000165
20.82723100000294	20.884897599993565	20.82335030000104
20.80391700000473	20.89037549999921	20.828929399998742
20.87521669999842	20.885131600000022	20.811529400001746
20.91481080000085	20.93468269999721	20.881663000000117
21.085760299996764	20.859826599997177	20.85081160000118
20.97296980000101	20.875606900001003	20.823041400006332
21.053496200001973	20.903108300000895	20.791531299997587
20.978821899996547	20.892179500006023	20.878361099996255
20.95425909999035	20.890548900002614	20.873186500000884
21.028453000006266	20.88354160000017	20.76938550000341

# C AES Implementering i python

## C.1 AES.py

```
# -----
# Imported libraries
# -----
from os.path import getsize
from os import remove
import numpy as np

# -----
# Fixed variables
# -----
# Sbox & inverse Sbox
subBytesTable = (
    0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b,
                           0xfe, 0xd7, 0xab, 0x76,
    0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf,
                           0x9c, 0xa4, 0x72, 0xc0,
    0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1,
                           0x71, 0xd8, 0x31, 0x15,
    0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2,
                           0xeb, 0x27, 0xb2, 0x75,
    0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3,
                           0x29, 0xe3, 0x2f, 0x84,
    0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39,
                           0x4a, 0x4c, 0x58, 0xcf,
    0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f,
                           0x50, 0x3c, 0x9f, 0xa8,
    0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21,
                           0x10, 0xff, 0xf3, 0xd2,
    0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d,
                           0x64, 0x5d, 0x19, 0x73,
    0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14,
                           0xde, 0x5e, 0x0b, 0xdb,
    0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62,
                           0x91, 0x95, 0xe4, 0x79,
    0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea,
                           0x65, 0x7a, 0xae, 0x08,
    0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f,
                           0x4b, 0xbd, 0x8b, 0x8a,
    0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9,
                           0x86, 0xc1, 0x1d, 0x9e,
    0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9,
                           0xce, 0x55, 0x28, 0xdf,
    0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f,
                           0xb0, 0x54, 0xbb, 0x16
)

invSubBytesTable = (
    0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38, 0xbf, 0x40, 0xa3, 0x9e,
                           0x81, 0xf3, 0xd7, 0xfb,
    0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f, 0xff, 0x87, 0x34, 0x8e, 0x43, 0x44,
                           0xc4, 0xde, 0xe9, 0xcb,
    0x54, 0x7b, 0x94, 0x32, 0xa6, 0xc2, 0x23, 0x3d, 0xee, 0x4c, 0x95, 0x0b,
                           0x42, 0xfa, 0xc3, 0x4e,
```

```

0x08, 0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2, 0x76, 0x5b, 0xa2, 0x49,
                           0x6d, 0x8b, 0xd1, 0x25,
0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xd4, 0xa4, 0x5c, 0xcc,
                           0x5d, 0x65, 0xb6, 0x92,
0x6c, 0x70, 0x48, 0x50, 0xfd, 0xed, 0xb9, 0xda, 0x5e, 0x15, 0x46, 0x57,
                           0xa7, 0x8d, 0x9d, 0x84,
0x90, 0xd8, 0xab, 0x00, 0x8c, 0xbc, 0xd3, 0x0a, 0xf7, 0xe4, 0x58, 0x05,
                           0xb8, 0xb3, 0x45, 0x06,
0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f, 0x0f, 0x02, 0xc1, 0xaf, 0xbd, 0x03,
                           0x01, 0x13, 0x8a, 0x6b,
0x3a, 0x91, 0x11, 0x41, 0x4f, 0x67, 0xdc, 0xea, 0x97, 0xf2, 0xcf, 0xce,
                           0xf0, 0xb4, 0xe6, 0x73,
0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85, 0xe2, 0xf9, 0x37, 0xe8,
                           0x1c, 0x75, 0xdf, 0x6e,
0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89, 0x6f, 0xb7, 0x62, 0x0e,
                           0xaa, 0x18, 0xbe, 0x1b,
0xfc, 0x56, 0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20, 0x9a, 0xdb, 0xc0, 0xfe,
                           0x78, 0xcd, 0x5a, 0xf4,
0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31, 0xb1, 0x12, 0x10, 0x59,
                           0x27, 0x80, 0xec, 0x5f,
0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0x0d, 0x2d, 0xe5, 0x7a, 0x9f,
                           0x93, 0xc9, 0x9c, 0xef,
0xa0, 0xe0, 0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0, 0xc8, 0xeb, 0xbb, 0x3c,
                           0x83, 0x53, 0x99, 0x61,
0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26, 0xe1, 0x69, 0x14, 0x63,
                           0x55, 0x21, 0x0c, 0x7d
)

# Round constants
round_constant = (
    0x00000000, 0x01000000, 0x02000000,
    0x04000000, 0x08000000, 0x10000000,
    0x20000000, 0x40000000, 0x80000000,
    0x1B000000, 0x36000000, 0x6C000000,
    0xD8000000, 0xAB000000, 0x4D000000,
)

# -----
# Main action functions
# -----
# Xtime
# Used to perform multiplication by x in the Galois field
def xtime(a):
    return (((a << 1) ^ 0x1B) & 0xFF) if (a & 0x80) else (a << 1)

# Byte substitution function
# Substitutes each byte in the state with a byte from the S-Box
def sub_bytes(data, bytesTable):
    for i, row in enumerate(data):
        for j, byte in enumerate(row):
            data[i][j] = bytesTable[byte]
    return data

# Shift rows function
# Shifts the rows of the matrix to the left.
# Each row is shifted by the number of its index

```

```

def shift_rows(array):
    array[:, 1] = np.roll(array[:, 1], -1, axis=0)
    array[:, 2] = np.roll(array[:, 2], -2, axis=0)
    array[:, 3] = np.roll(array[:, 3], -3, axis=0)
    return array

# Inverse shift rows function
# Shifts the rows of the matrix to the right.
# Each row is shifted by the number of its index
def inv_shift_rows(array):
    array[:, 1] = np.roll(array[:, 1], 1, axis=0)
    array[:, 2] = np.roll(array[:, 2], 2, axis=0)
    array[:, 3] = np.roll(array[:, 3], 3, axis=0)
    return array

# Performs the mix columns layer
def mix_columns(data):
    # mixes a single column
    def mix_single_column(data):
        t = data[0] ^ data[1] ^ data[2] ^ data[3]
        u = data[0]
        data[0] ^= t ^ xtime(data[0] ^ data[1])
        data[1] ^= t ^ xtime(data[1] ^ data[2])
        data[2] ^= t ^ xtime(data[2] ^ data[3])
        data[3] ^= t ^ xtime(data[3] ^ u)

    # mixes all columns using mix_single_column
    def mix(data):
        for i in range(4):
            mix_single_column(data[i])
        return data
    data = mix(data)
    return data

# Preforms the inverse mix columns layer
# This function is similar to the mix_columns function
# but instead preforms the inverse operation.
def inv_mix_columns(data):
    for i in range(4):
        u = xtime(xtime(data[i][0] ^ data[i][2]))
        v = xtime(xtime(data[i][1] ^ data[i][3]))
        data[i][0] ^= u
        data[i][1] ^= v
        data[i][2] ^= u
        data[i][3] ^= v
    mix_columns(data)
    return data

# Adds a padding to ensure a bloke size of 16 bytes
def add_padding(data):
    length = 16 - len(data)
    for i in range(length):
        data.append(0)
    return data, length

```

```

# Removes the padding from the data
def remove_padding(data, identifier):
    if identifier[-1] == 0:
        return data
    elif identifier[-1] > 0 and identifier[-1] < 16:
        return data[:-identifier[-1]]
    else:
        raise ValueError('Invalid padding')

# Performs the encryption rounds on the input data matrix
# This function is used for the encryption of data matrixes
# using the expanded keys.
def encryption_rounds(data, round_keys, nr):
    # Inizial add round key
    data = np.bitwise_xor(data, round_keys[0])

    # Rounds 1 to 9 or 1 to 11 or 1 to 13
    # Here each step in one round is performed in a sequence n times
    # where n is the number of rounds minus the last round.
    for i in range(1, (nr - 1)):
        # Sub bytes
        data = sub_bytes(data, subBytesTable)
        # Shift rows
        data = shift_rows(data)
        # Mix columns
        data = mix_columns(data)
        # Add round key
        data = np.bitwise_xor(data, round_keys[i])

    # Final round
    # Identical to the previous rounds, but without mix columns
    data = sub_bytes(data, subBytesTable)
    data = shift_rows(data)
    data = np.bitwise_xor(data, round_keys[nr - 1])

    # Returns the encrypted data
    return data

# Performs the decryption rounds on the input data matrix
# This function is used for the decryption of data matrixes
# using the expanded keys.
def decryption_rounds(data, round_keys, nr):
    # Inizial add round key, inverse shift rows and inverse sub bytes
    data = np.bitwise_xor(data, round_keys[-1])
    data = inv_shift_rows(data)
    data = sub_bytes(data, invSubBytesTable)

    # Rounds 1 to 9 or 1 to 11 or 1 to 13
    # Here each step in one round is performed in a sequence n times
    # where n is the number of rounds minus the last round.
    for i in range(1, (nr - 1)):
        # Add round key
        data = np.bitwise_xor(data, round_keys[-(i+1)])
        # Inverse mix columns
        data = inv_mix_columns(data)
        # Inverse shift rows

```

```

        data = inv_shift_rows(data)
        # Inverse sub bytes
        data = sub_bytes(data, invSubBytesTable)

    # Final round
    # Final add round key of final round
    data = np.bitwise_xor(data, round_keys[0])

    # Returns the decrypted data
    return data

# -----
# Key expansion setup
# -----
# Key expansion function
# This function is used to expand the key to the correct number of round
# keys for the encryption and decryption rounds.
def keyExpansion(key):
    # Format key correctly for the key expansion
    key = [key[i:i+2] for i in range(0, len(key), 2)]

    # Key expansion setup
    # This part determines the number of rounds and the number of words
    # using the key length.
    if len(key) == 16:
        words = key_schedule(key, 4, 11)
        nr = 11
    if len(key) == 24:
        words = key_schedule(key, 6, 13)
        nr = 13
    if len(key) == 32:
        words = key_schedule(key, 8, 15)
        nr = 15

    # Create list for storing the round keys & tmp list for storing
    # for temporary storage.
    round_keys = [None for i in range(nr)]
    tmp = [None for i in range(4)]

    # Formats the words to a list of tuples
    for i in range(nr * 4):
        for index, t in enumerate(words[i]):
            tmp[index] = int(t, 16) # type: ignore
        words[i] = tuple(tmp)

    # Formats teh words to a list of numpy arrays where each
    # array is a 4x4 matrix representing a round key.
    for i in range(nr):
        round_keys[i] = np.array(words[i * 4] + words[i * 4 + 1] + words[i * 4 + 2] + words[i * 4 + 3]).reshape(4, 4) # type: ignore

    # Returns the list of round keys and the number of rounds
    return round_keys, nr

# Key schedule (nk = number of columns, nr = number of rounds)
# This function is used to expand the key to the correct number of round

```

```

def key_schedule(key, nk, nr):
    # Create list for storing the words and populates the first
    # 4 with the specified key.
    words = [(key[4*i], key[4*i+1], key[4*i+2], key[4*i+3]) for i in range(
        nk)]

    # Fill out the rest based on previous four words using the functions,
    # rotword,
    # subword and rcon values
    limit = False
    for i in range(nk, (nr * nk)):
        # Get required previous keywords
        temp, word = words[i-1], words[i-nk]

        # If multiple of nk use rot, sub, rcon etc
        if i % nk == 0:
            x = SubWord(RotWord(temp))
            rcon = round_constant[int(i/nk)]
            temp = hexor(x, hex(rcon)[2:])
            limit = False
        elif i % 4 == 0:
            limit = True

        if i % 4 == 0 and limit and nk >= 8:
            temp = SubWord(temp)

        # Xor the two hex values
        xord = hexor(''.join(word), ''.join(temp))
        # Add to list
        words.append((xord[:2], xord[2:4], xord[4:6], xord[6:8]))
    # Return the list of words
    return words

# Takes two hex values and calculates hex1 xor hex2
def hexor(hex1, hex2):
    # Convert to binary
    bin1 = hex2binary(hex1)
    bin2 = hex2binary(hex2)

    # Calculate
    xord = int(bin1, 2) ^ int(bin2, 2)

    # Cut prefix
    hexed = hex(xord)[2:]

    # Leading 0s get cut above, if not length 8 add a leading 0
    if len(hexed) != 8:
        hexed = '0' + hexed

    # Return hex
    return hexed

# Takes a hex value and returns binary
def hex2binary(hex):
    return bin(int(str(hex), 16))

```

```

# Takes from 1 to the end, adds on from the start to 1
def RotWord(word):
    return word[1:] + word[:1]

# Selects correct values from sbox based on the current word
# and replaces the word with the new values.
def SubWord(word):
    # Create list for storing the new word
    sWord = []

    # Loop through the current word
    for i in range(4):

        # Check first char, if its a letter(a-f) get corresponding decimal
        # otherwise just take the value and add 1
        if word[i][0].isdigit() is False:
            row = ord(word[i][0]) - 86
        else:
            row = int(word[i][0])+1

        # Repeat above for the second char
        if word[i][1].isdigit() is False:
            col = ord(word[i][1]) - 86
        else:
            col = int(word[i][1])+1

        # Get the index base on row and col (16x16 grid)
        sBoxIndex = (row*16) - (17-col)

        # Get the value from sbox and removes prefix (0x)
        piece = hex(subBytesTable[sBoxIndex])[2:]

        # Check length to ensure leading 0s are not forgotten
        if len(piece) != 2:
            piece = '0' + piece

        # Adds the new value to the list
        sWord.append(piece)

    # Returning word as string
    return ''.join(sWord)

# -----
# Running modes setup
# -----
# ECB encryption function
def ecb_enc(key, file_path):
    file_size = getSize(file_path)
    round_keys, nr = keyExpansion(key)

    with open(f"{file_path}.enc", 'wb') as output, open(file_path, 'rb') as data:
        for i in range(int(file_size/16)):
            raw = np.array([i for i in data.read(16)]).reshape(4, 4)
            result = bytes((encryption_rounds(raw, round_keys, nr).flatten()
                           .tolist()))
            output.write(result)

```

```

if file_size % 16 != 0:
    raw = [i for i in data.read()] # type: ignore
    raw, length = add_padding(raw)

    result = bytes((encryption_rounds(np.array(raw).reshape(4, 4),
                                      round_keys, nr).flatten()
                           .tolist()))
    identifier = bytes((encryption_rounds(np.array([0 for i in range
                                                    (15)] + [length]).reshape(
                                              4, 4), round_keys, nr).
                           flatten()).tolist())

    output.write(result + identifier)
else:
    identifier = bytes((encryption_rounds(np.array([0 for i in range
                                                    (16)]).reshape(4, 4),
                                           round_keys, nr).flatten()
                           .tolist()))

    output.write(identifier)
remove(file_path)

# ECB decryption function
def ecb_dec(key, file_path):
    file_size = getsize(file_path)
    file_name = file_path[:-4]
    round_keys, nr = keyExpansion(key)

    with open(f"{file_name}", 'wb') as output, open(file_path, 'rb') as data
        :
            for i in range(int(file_size/16) - 2):
                raw = np.array([i for i in data.read(16)]).reshape(4, 4)
                result = bytes((decryption_rounds(raw, round_keys, nr).flatten()
                               .tolist()))
                output.write(result)

                data_pice = np.array([i for i in data.read(16)]).reshape(4, 4)
                identifier = np.array([i for i in data.read()]).reshape(4, 4)

                result = (decryption_rounds(data_pice, round_keys, nr).flatten()
                          .tolist())
                identifier = (decryption_rounds(identifier, round_keys, nr).flatten()
                              .tolist())

                result = bytes(remove_padding(result, identifier))

                output.write(result)
            remove(file_path)

# CBC encryption function
def cbc_enc(key, file_path, iv):
    file_size = getsize(file_path)
    vector = np.array([int(iv[i:i+2], 16) for i in range(0, len(iv), 2)]).
                           reshape(4, 4)
    round_keys, nr = keyExpansion(key)

```

```

with open(f"{file_path}.enc", 'wb') as output, open(file_path, 'rb') as data:
    for i in range(int(file_size/16)):
        raw = np.array([i for i in data.read(16)]).reshape(4, 4)
        raw = np.bitwise_xor(raw, vector)
        vector = encryption_rounds(raw, round_keys, nr)
        output.write(bytes((vector.flatten()).tolist()))

    if file_size % 16 != 0:
        raw = [i for i in data.read()] # type: ignore
        raw, length = add_padding(raw)

        raw = np.bitwise_xor(np.array(raw).reshape(4, 4), vector)
        vector = encryption_rounds(raw, round_keys, nr)

        identifier = np.bitwise_xor(np.array([0 for i in range(15)] + [
            length]).reshape(4, 4), vector)
        identifier = encryption_rounds(identifier, round_keys, nr)

        output.write(bytes((vector.flatten()).tolist() + (identifier.
            flatten()).tolist()))
    else:
        identifier = np.bitwise_xor(np.array([0 for i in range(16)]).
            reshape(4, 4), vector)
        identifier = bytes(((encryption_rounds(identifier, round_keys,
            nr)).flatten()).tolist()) # type: ignore
        output.write(identifier) # type: ignore
remove(file_path)

# CBC decryption function
def cbc_dec(key, file_path, iv):
    iv = np.array([int(iv[i:i+2], 16) for i in range(0, len(iv), 2)]).
        reshape(4, 4)
    file_size = getsize(file_path)
    file_name = file_path[:-4]
    round_keys, nr = keyExpansion(key)

    with open(f"{file_name}", 'wb') as output, open(file_path, 'rb') as data:
        :
        if int(file_size/16) - 3 >= 0:
            vector = np.array([i for i in data.read(16)]).reshape(4, 4)
            raw = decryption_rounds(vector, round_keys, nr)
            result = np.bitwise_xor(raw, iv)
            output.write(bytes((result.flatten()).tolist()))

            for i in range(int(file_size/16) - 3):
                raw = np.array([i for i in data.read(16)]).reshape(4, 4)
                result = decryption_rounds(raw, round_keys, nr)
                result = np.bitwise_xor(result, vector)
                vector = raw
                output.write(bytes((result.flatten()).tolist()))
        else:
            vector = iv

    data_pice = np.array([i for i in data.read(16)]).reshape(4, 4)

```

```

vector_1, identifier = data_pice, np.array([i for i in data.read()]).  

                      .reshape(4, 4)

result = decryption_rounds(data_pice, round_keys, nr)
identifier = decryption_rounds(identifier, round_keys, nr)

identifier = np.bitwise_xor(identifier, vector_1)
data_pice = np.bitwise_xor(result, vector)

result = bytes(remove_padding((data_pice.flatten()).tolist(), (  

                                         identifier.flatten()).tolist()  

                                     ))

output.write(result)
remove(file_path)

# PCBC encryption function
def pcbc_enc(key, file_path, iv):
    file_size = getsize(file_path)
    vector = np.array([int(iv[i:i+2], 16) for i in range(0, len(iv), 2)]).  

                           reshape(4, 4)
    round_keys, nr = keyExpansion(key)

    with open(f"{file_path}.enc", 'wb') as output, open(file_path, 'rb') as  

        data:
        for i in range(int(file_size/16)):
            raw = np.array([i for i in data.read(16)]).reshape(4, 4)
            tmp = np.bitwise_xor(raw, vector)
            vector = encryption_rounds(tmp, round_keys, nr)
            output.write(bytes((vector.flatten()).tolist()))
            vector = np.bitwise_xor(vector, raw)

            if file_size % 16 != 0:
                raw = [i for i in data.read()] # type: ignore
                raw, length = add_padding(raw)
                raw = np.array(raw).reshape(4, 4)

                tmp = np.bitwise_xor(raw, vector)
                vector1 = encryption_rounds(tmp, round_keys, nr)
                vector = np.bitwise_xor(vector1, raw)

                identifier = np.bitwise_xor(np.array([0 for i in range(15)] + [  

                                                               length]).reshape(4, 4),  

                                           vector)
                identifier = encryption_rounds(identifier, round_keys, nr)

                output.write(bytes((vector1.flatten()).tolist() + (identifier.  

                                                               flatten()).tolist()))

            else:
                identifier = np.bitwise_xor(np.array([0 for i in range(16)]).  

                                         reshape(4, 4), vector)
                identifier = bytes((encryption_rounds(identifier, round_keys, nr)  

                                   ).flatten().tolist()) #  

                                         type: ignore
                output.write(identifier) # type: ignore
remove(file_path)

```

```

# PCBC decryption function
def pcbc_dec(key, file_path, iv):
    iv = np.array([int(iv[i:i+2], 16) for i in range(0, len(iv), 2)]).
        reshape(4, 4)
    file_size = getsize(file_path)
    file_name = file_path[:-4]
    round_keys, nr = keyExpansion(key)

    with open(f"{file_name}", 'wb') as output, open(file_path, 'rb') as data
        :
            if int(file_size/16) - 3 >= 0:
                vector = np.array([i for i in data.read(16)]).reshape(4, 4)
                raw = decryption_rounds(vector, round_keys, nr)
                result = np.bitwise_xor(raw, iv)
                vector = np.bitwise_xor(vector, result)
                output.write(bytes((result.flatten()).tolist()))

                for i in range(int(file_size/16) - 3):
                    raw = np.array([i for i in data.read(16)]).reshape(4, 4)
                    result = decryption_rounds(raw, round_keys, nr)
                    result = np.bitwise_xor(result, vector)
                    vector = np.bitwise_xor(raw, result)
                    output.write(bytes((result.flatten()).tolist()))
            else:
                vector = iv

            data_pice = np.array([i for i in data.read(16)]).reshape(4, 4)
            vector_1, identifier = data_pice, np.array([i for i in data.read()]).
                reshape(4, 4)

            result = decryption_rounds(data_pice, round_keys, nr)
            data_pice = np.bitwise_xor(result, vector)

            vector_1 = np.bitwise_xor(vector_1, data_pice)
            identifier = decryption_rounds(identifier, round_keys, nr)
            identifier = np.bitwise_xor(identifier, vector_1)

            result = bytes(remove_padding((data_pice.flatten()).tolist(), (
                identifier.flatten()).tolist()))
        output.write(result)
    remove(file_path)

# OFB encryption function
def ofb_enc(key, file_path, iv):
    file_size = getsize(file_path)
    round_keys, nr = keyExpansion(key)
    mix = np.array([int(iv[i:i+2], 16) for i in range(0, len(iv), 2)]).
        reshape(4, 4)
    iv = mix

    with open(f"{file_path}.enc", 'wb') as output, open(file_path, 'rb') as data:
        for i in range(int(file_size/16)):
            raw = np.array([i for i in data.read(16)]).reshape(4, 4)
            mix = encryption_rounds(mix, round_keys, nr)
            result = np.bitwise_xor(raw, mix)

```

```

        output.write(bytes((result.flatten()).tolist()))

    if file_size % 16 != 0:
        raw = [i for i in data.read()] # type: ignore
        length = add_padding(raw)
        raw = np.array(raw).reshape(4, 4)

        if file_size < 16:
            mix = encryption_rounds(iv, round_keys, nr)
        else:
            mix = encryption_rounds(mix, round_keys, nr)
    result = np.bitwise_xor(mix, raw)

    mix = encryption_rounds(mix, round_keys, nr)
    identifier = np.bitwise_xor(np.array([0 for i in range(15)] + [
                                length]).reshape(4, 4),
                                mix)

    output.write(bytes((result.flatten()).tolist() + (identifier.
                                                    flatten()).tolist()))

else:
    mix = encryption_rounds(mix, round_keys, nr)
    identifier = np.bitwise_xor(np.array([0 for i in range(16)]).
                                reshape(4, 4), mix)
    output.write(bytes(identifier.flatten()).tolist())
remove(file_path)

# OFB decryption function
def ofb_dec(key, file_path, iv):
    iv = np.array([int(iv[i:i+2], 16) for i in range(0, len(iv), 2)]).
        reshape(4, 4)
    file_size = getsize(file_path)
    file_name = file_path[:-4]
    round_keys, nr = keyExpansion(key)

    with open(f"{file_name}", 'wb') as output, open(file_path, 'rb') as data
        :
            if int(file_size/16) - 3 >= 0:
                raw = np.array([i for i in data.read(16)]).reshape(4, 4)
                mix = encryption_rounds(iv, round_keys, nr)
                result = np.bitwise_xor(raw, mix)
                output.write(bytes((result.flatten()).tolist()))

                for i in range(int(file_size/16) - 3):
                    raw = np.array([i for i in data.read(16)]).reshape(4, 4)
                    mix = encryption_rounds(mix, round_keys, nr)
                    result = np.bitwise_xor(raw, mix)
                    output.write(bytes((result.flatten()).tolist()))
            else:
                mix = iv

            data_pice = np.array([i for i in data.read(16)]).reshape(4, 4)
            identifier = np.array([i for i in data.read()]).reshape(4, 4)

            mix = encryption_rounds(mix, round_keys, nr)
            data_pice = np.bitwise_xor(data_pice, mix)

            mix = encryption_rounds(mix, round_keys, nr)

```

## C.2. ENCRYPT.PY

---

```
identifier = np.bitwise_xor(identifier, mix)

result = bytes(remove_padding((data_pice.flatten()).tolist(), (
    identifier.flatten().tolist()
)) # type: ignore

output.write(result) # type: ignore
remove(file_path)
```

## C.2 encrypt.py

```
from PyAES import AES
from sys import argv


# -----
# Encryption function
# -----
def encrypt(key, file_path, running_mode, iv=None):

    # Input validation
    if (len(key) / 2) not in [16, 24, 32]:
        raise Exception('Key length is not valid')
    elif running_mode in ["CBC", "PCBC", "CFB", "OFB", "CTR", "GCM"]:
        if (len(iv) / 2) != 16 or iv is None:
            raise Exception('IV length is not valid')

    # Running mode selection
    if running_mode == "ECB":
        AES.ecb_enc(key, file_path)
    elif running_mode == "CBC" and iv is not None:
        AES.cbc_enc(key, file_path, iv)
    elif running_mode == "PCBC" and iv is not None:
        AES.pcbc_enc(key, file_path, iv)
    elif running_mode == "OFB" and iv is not None:
        AES.ofb_enc(key, file_path, iv)
    else:
        raise Exception("Running mode not supported")

if __name__ == "__main__":
    encrypt(key=argv[1], file_path=argv[2], running_mode=argv[3], iv=argv[4]
            )
```

## C.3 decrypt.py

```
from PyAES import AES
from sys import argv


# -----
# Decryption function
# -----
def decrypt(key, file_path, running_mode, iv=None):
```

```
# Input validation
if file_path[-4:] != ".enc":
    raise Exception('File is not encrypted in known format')
if (len(key) / 2) not in [16, 24, 32]:
    raise Exception('Key length is not valid')
elif running_mode in ["CBC", "PCBC", "CFB", "OFB", "CTR", "GCM"]:
    if (len(iv) / 2) != 16 or iv is None:
        raise Exception('IV length is not valid')

# Running mode selection
if running_mode == "ECB":
    AES.ecb_dec(key, file_path)
elif running_mode == "CBC" and iv is not None:
    AES.cbc_dec(key, file_path, iv)
elif running_mode == "PCBC" and iv is not None:
    AES.pcbc_dec(key, file_path, iv)
elif running_mode == "OFB" and iv is not None:
    AES.ofb_dec(key, file_path, iv)
else:
    raise Exception("Running mode not supported")

if __name__ == "__main__":
    decrypt(key=argv[1], file_path=argv[2], running_mode=argv[3], iv=argv[4])
```

## C.4 \_\_main\_\_.py

```

print("""This is a simple AES (Advanced Encryption Standard)
implementation in Python-3. It is
a pure Python implementation of AES that is designed to be used as a
educational tool
only. It is not intended to be used in any other use case than educational
and no
security is guaranteed for data encrypted or decrypted using this tool.""")
print("-"*85)
run()

def run():
    action = input("Do you want to encrypt, decrypt or quit? (e/d/q): ")
    if action == "e":
        running_mode = input("Please select cipher running mode (ECB/CBC/
PCBC/CFB/OFB/CTR/GCM): ")

        if running_mode == "ECB":
            key = getpass(prompt="Please enter your key: ")
            file_path = input("Please enter path to file: ")
            confirmation = input("Are you sure you want to encrypt this file
? (y/n): ")

            if confirmation == "y":
                encrypt(key, file_path, running_mode)
                print("\nEncryption complete!")

            elif confirmation == "n":
                print("Encryption aborted!")
                exit()

            else:
                print("Invalid input!")
                exit()

        elif running_mode in ["CBC", "PCBC", "CFB", "OFB", "CTR", "GCM"]:
            key = getpass(prompt="Please enter your key: ")
            iv = getpass(prompt="Please enter your iv: ")
            file_path = input("Please enter path to file: ")
            confirmation = input("Are you sure you want to encrypt this file
? (y/n): ")

            if confirmation == "y":
                encrypt(key, file_path, running_mode, iv)
                print("\nEncryption complete!")

            elif confirmation == "n":
                print("Encryption aborted!")
                exit()

            else:
                print("Invalid input!")
                exit()

        else:
            print("Invalid cipher running mode")
            run()

    elif action == "d":

```

```

running_mode = input("Please select cipher running mode (ECB/CBC/
PCBC/CFB/OFB/CTR/GCM): ")

if running_mode == "ECB":
    key = getpass(prompt="Please enter your key: ")
    file_path = input("Please enter path to file: ")
    confirmation = input("Are you sure you want to decrypt this file
? (y/n): ")

    if confirmation == "y":
        decrypt(key, file_path, running_mode)
        print("\nDecryption complete!")

    elif confirmation == "n":
        print("Decryption aborted!")
        exit()

    else:
        print("Invalid input!")
        exit()

elif running_mode in ["CBC", "PCBC", "CFB", "OFB", "CTR", "GCM"]:
    key = getpass(prompt="Please enter your key: ")
    iv = getpass(prompt="Please enter your iv: ")
    file_path = input("Please enter path to file: ")
    confirmation = input("Are you sure you want to decrypt this file
? (y/n): ")

    if confirmation == "y":
        decrypt(key, file_path, running_mode, iv)
        print("\nDecryption complete!")

    elif confirmation == "n":
        print("Decryption aborted!")
        exit()

    else:
        print("Invalid input!")
        exit()

else:
    print("Invalid cipher running mode")
    run()

elif action == "q":
    print("Exiting...")
    exit()

else:
    print("Invalid action (to quit enter 'q')")
    run()

if __name__ == "__main__":
    main()

```

## D Test kod (Analyze.py)

```
# -----  
#           Background structure  
# -----  
  
# Imports the encrypt function from the PyAES module  
from PyAES.encrypt import encrypt  
# Imports the timer and random number generator  
from time import perf_counter  
from random import randint  
# Imports the remove function for deleting files  
from os import remove  
  
  
# For displaying progress during test runs  
def progress_bar(progress, total_progress):  
    percent = 100 * (float(progress) / float(total_progress))  
    bar_progress = int(100 * (float(progress) / float(total_progress)))  
  
    if bar_progress > 100 or percent > 100:  
        bar_progress = 100  
        percent = 100  
  
    bar_remaining = 100 - bar_progress  
    bar = '#' * bar_progress + '-' * bar_remaining  
    print(f"\r[{bar}] {percent:.2f}%", end="\r")  
    return progress + 1  
  
  
# Writes the resulting data to a text file  
def write_data(data, name_f):  
    # Creates a text file with the specified name  
    with open(name_f + '.txt', 'w') as f:  
        for i in data:  
            # Writes the data to the text file  
            f.write(str(i) + '\n')  
    f.write('\n')  
  
  
# Creates a text file with specified size and fills it with random bytes  
def setup(count):  
    # Creates a text file with the name test_speed.txt  
    with open("test_speed.txt", 'wb') as f:  
        for j in range(count):  
            # Writes random bytes to the text file  
            f.write(bytes([randint(0, 255)]))  
  
  
# Runs the specified function and returns the time it takes to run  
def speed_test(count, key, mode, iv):  
    # Creates a text file with the specified size  
    setup(count)  
    # Starts the timer  
    start = perf_counter()  
    # Executes the function  
    encrypt(key, "test_speed.txt", mode, iv=iv)  
    # Stops the timer  
    end = perf_counter()
```

```

# Deletes the text file
remove("test_speed.txt.enc")
return end - start

# Executes the code if the file is run directly
if __name__ == '__main__':
    # Test parameters
    keys = ["2b7e151628aed2a6abf7158809cf4f3c",
            "8e73b0f7da0e6452c810f32b809079e562f8ead2522c6b7b",
            "603deb1015ca71be2b73aef0857d77811f352c073b6108d72d9810a30914dff4"]
    iv = "000102030405060708090a0b0c0d0e0f"
    file_size = 1000000
    runs = 25

# -----
#     Test time difference between 128, 192 and 256 bit keys
# -----
data = []
progress = 0
# Loops through the different key sizes
for i in keys:
    data_tmp = []
    # Runs the test the specified amount of times
    for j in range(runs):
        # Displays the progress
        progress = progress_bar(progress, 150)
        # Runs the test and saves the result
        test = speed_test(file_size, i, 'ECB', iv)
        # Saves the result
        data_tmp.append(test)
    # Writes the results of every run to a text file
    write_data(data_tmp, 'keys_test_raw ' + str(len(i) * 4))
    # Saves the average of the results
    data.append(sum(data_tmp)/len(data_tmp))
# Writes the average of the results to a text file
write_data(data, 'keys_test')

# -----
#     Test time difference between ECB, CBC and OFB modes
# -----
data = []
data_tmp = []

# Runs the time test for OFB the specified amount of times
for i in range(runs):
    # Displays the progress
    progress = progress_bar(progress, 150)
    # Runs the test for OFB
    test = speed_test(file_size, keys[0], 'OFB', iv)
    # Saves the result in a temporary list
    data_tmp.append(test)
# Writes the results of every run to a text file
write_data(data_tmp, 'modes_test_raw OFB')
# Saves the average of the results in a list and clears the temporary
# list
data.append(sum(data_tmp)/len(data_tmp))
data_tmp = []

# Runs the time test for CBC the specified amount of times

```

---

```

for i in range(runs):
    # Displays the progress
    progress = progress_bar(progress, 150)
    # Runs the test for CBC
    test = speed_test(file_size, keys[0], 'CBC', iv)
    # Saves the result in a temporary list
    data_tmp.append(test)
# Writes the the results of every run to a text file
write_data(data_tmp, 'modes_test_raw CBC')
# Saves the average of the results in a list and clears the temporary
# list
data.append(sum(data_tmp)/len(data_tmp))
data_tmp = []

# Runs the time test for ECB the specified amount of times
for i in range(runs):
    # Displays the progress
    progress = progress_bar(progress, 150)
    # Runs the test for ECB
    test = speed_test(file_size, keys[0], 'ECB', iv)
    # Saves the result in a temporary list
    data_tmp.append(test)
# Writes the the results of every run to a text file
write_data(data_tmp, 'modes_test_raw ECB')
# Saves the average of the results in a list
data.append(sum(data_tmp)/len(data_tmp))
# Writes the averages from every run to a text file
write_data(data, 'modes_test')

# Dilsapys the progress finished
progress = progress_bar(progress, 150)
# Prints that the tests is completed
print("\n")
print("Completed")

```