

The AES encryption algorithm

En undersökning av The Advanced Encryption Standard (AES)

Klass:

NA20

Handledare:

Jimmy Nylén

Författare:

Gabriel Lindeblad

Program:

Naturvetenskapsprogrammet

Abstract

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut lacinia ex eget sagittis congue. Nullam cursus egestas dolor, suscipit gravida magna ultrices sit amet. Nullam placerat dui eu arcu pharetra, sit amet tempor dolor convallis. Aenean sodales condimentum turpis, commodo maximus augue. Aenean vel nibh dui. Pellentesque ex libero, lacinia nec mauris vel, convallis consectetur felis. Maecenas ut nibh sed magna maximus imperdiet at id purus. In vel consequat metus. Donec non tincidunt nunc. Sed pulvinar odio ut sapien vestibulum, quis mollis arcu tempor. Maecenas ut sem leo. Sed leo risus, mollis eu ex vitae, feugiat consequat metus. Aenean interdum volutpat urna, nec tempor mi accumsan quis. Morbi blandit maximus urna non aliquet aes.

Innehåll

Ordlista	5
Akronymer	9
1 Inledning	10
1.1 Syfte	10
1.2 Frågeställningar	10
1.3 Avgränsning	10
2 Bakgrund	11
2.1 Kryptografi	11
2.1.1 Uppkomst	11
2.1.2 Utveckling	11
2.2 AES Uppkomst	12
2.3 Tidigare forskning	12
3 Teori	13
3.1 Kryptering	13
3.2 Blockskiffer	13
3.2.1 Körlägen	13
3.2.1.1 ECB	14
3.2.1.2 CBC	15
3.2.1.3 OFB	17
3.3 Symetrisk & Asymmetrisk Kryptering	18
3.4 AES	18
3.4.1 Finite Fields	19
3.4.2 AES S-Box	19
3.4.3 Struktur	20
3.4.3.1 SubBytes operationen	22
3.4.3.2 ShiftRows operationen	23
3.4.3.3 Inverse ShiftRows operationen	23
3.4.3.4 MixColumns operationen	23
3.4.3.5 Inverse MixColumns operationen	24
3.4.3.6 AddRoundKey operationen	24
3.4.4 Nyckel utökning	24
3.4.4.1 RotWord	24
3.4.4.2 SubWord	24
3.4.4.3 Rcon	24
3.4.4.4 Nyckel utökning 128bit	24
3.4.4.5 Nyckel utökning 192bit	24
3.4.4.6 Nyckel utökning 256bit	24
3.4.5 AES-128bit	24
3.4.6 AES-192bit	24
3.4.7 AES-256bit	24
4 Metod & Genomförande	25
4.1 Implementering	25

4.2	Test Uppsättning	25
4.2.1	Nyckellängds Test	25
4.2.2	Körläges Test	25
4.2.3	Krypterings Test	25
4.3	Genomförande	25
5	Resultat	26
5.1	Nyckellängds Test	26
5.2	Körläges Test	26
5.3	Krypterings Test	26
6	Diskussion & Slutord	27
6.1	Felkällor	27
6.2	Förbättringar	27
6.3	Slutsats	27
6.4	Slutord	27
	Källförteckning	28
A	AES körläges test resultat	31
A.1	Före testet	31
A.2	Efter ECB kryptering	32
A.3	Efter CBC kryptering	32
A.4	Efter CFB kryptering	33
B	Rå data från test	34
B.1	Data från Nyckellängds test	34
B.2	Data från körläges test	35
C	AES Implementering i python	36
C.1	AES.py	36
C.2	encrypt.py	47
C.3	decrypt.py	48
C.4	__main__.py	48
D	Test kod (Analyze.py)	52

Figur lista

3.1	Electronic Code Book läge kryptering [Wik22c]	14
3.2	Electronic Code Book läge dekryptering [Wik22d]	15
3.3	Cipher Block Chaining läge kryptering [Wik22e]	16
3.4	Cipher Block Chaining läge dekryptering [Wik22f]	16
3.5	Output Feedback läge kryptering [Wik22g]	17
3.6	Output Feedback läge dekryptering [Wik22h]	17
3.7	AES S-box (Den vänstra matrisen) & Invers S-box (Den högra matrisen) [Wik22r]	20
3.8	Vanlig runda [Wik99]	21
3.9	Krypterings runda funktion	22
3.10	Dekrypterings runda funktion	22
3.11	SubBytes operationen [Wik21a]	23
3.12	ShiftRows operationen [Wik20a]	23
3.13	MixColumns operationen [Wik20b]	23
5.1	AES krypterings test (ECB, CBC, OFB)	26
A.1	Orginal bild	31
A.2	Efter ECB Kryptering	32
A.3	Efter CBC Kryptering	32
A.4	Efter OFB Kryptering	33

Ordlista

A | B | C | E | F | H | N | O | P | R | S | V | 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.[[Wik22n](#)]

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 abstract 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 däremot 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 medelandet med en annan. Ersättnings 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](#)]

E

Enigma

Enigma var ett krypterings verktyg som användes under andra världskriget av tyska milisen för att kryptera medelanden. Maskinen bestod av en elektromekanisk rotordisk som under tiden medelandet skrivas in för kryptering även ändrar de 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 de tidiga 1900-talet.[[Wik22j](#)]

F

Frekvensanalys

Frekvensanalys inom kryptografi är en metod för att knäcka ett Substitutionsskiffer genom att analysera frekvensen av bokstäver och utnyttja de 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 medelandet.[[Wik21](#)]

H

Hashfunktion

Hashfunktion är en funktion som man kan säga delar upp en viss datamängd och genom för en serie operationer som resulterar i hashtext av godkänd längd. Längden är samma för alla hastexter som använder samma funktion medan innehållet förändras så fort något över huvud taget ändras i datan som funktionen appliceras på. Användningsområdet för dessa funktioner är bland annat när man vill kunna verifiera medelanden eller information och försäkra sig om att ingen ändrat på medelandet efter att de skickats. Detta kan man då göra för att man vet att om man skör informationen genom samma hashfunktion borde resultatet vara identisk ifall informationen är oförändrad.[[Wik22m](#)]

HTTP

[[BLFF96](#)]

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.[[Wik22n](#)]

Nyckelström

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

O

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.[[Wik22n](#)]

P

Polyalphabetic substitutionsskiffer

Polyalphabetic substitutionsskiffer bygger på att man använder flera olika Substitutionsskiffer för att på så sätt undvika en ut av de 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.[[Wik22o](#)]

Pseudoslump

Pseudoslump är en rad av nummer som kan se ut att vara helt slumpmässiga men har blivit framställda genom en upprepbar process.[[Wik22p](#)]

Python

Python är ett högnivå programmerings språk byggt på programmerings språket C. De är skapat av Guido van Rossum och släpptes i Februari 1991.[[Pyt22b](#)]

R

RSA

Rivest-Shamir-Adleman (RSA) är en av de mest välkända krypteringsalgoritmerna och var en av de första algorithmerna som byggde på en asymetrisk kryptering. RSA bygger på multiplikation av stora primtal där primtalen är nycklarna.[[Wik22q](#)]

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 medelande. 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.[[Wik22u](#)]

SSH

[[Bar+01](#)]

Strömskiffer

Strömskiffer, ett symetriskt nyckel skiffer där man använder en Pseudoslumpmässig skiffer ström (Nyckelström) som sedan en Bit i taget kombineras med de som ska krypteras. Den kombinerande operationen som används i strömskiffer är ofta en XOR-operation.[[Wik22s](#)]

Substitutionsskiffer

Ett Substitutionsskiffer är en typ av krypterings metod 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.[[Wik22t](#)]

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](#)]

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

AES	Advanced Encryption Standard
ASCII	American Standard Code for Information Interchange
CBC	Cipher Block Chaining läge
DES	Data Encryption Standard
ECB	Electronic Code Book läge
IV	Initialization Vector
OFB	Output Feedback läge
SSL	Secure Socket Layer
TLS	Transport Layer Security
WPA2	Wi-Fi Protected Access 2

1 Inledning

Kryptering, en bärande grundsten i dagens digitaliserade samhälle. De 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 exempelvis fanns redan för 2000 år sedan när Julius Caesar använde de vi idag kallar Caesarchiffer för att skicka hemliga meddelanden.¹

Sedan dess har kryptografi självklart utvecklats enormt och vi har gått från de på ett sätt enkla men även eleganta Caesarchiffer som användes då till moderna algoritmer såsom Advanced Encryption Standard och Data Encryption Standard. Dessa algoritmer har samma syfte som Caesarchiffer men har utvecklats under en tid där datorer står som de dominerande informationshanteringsverktyget, vilket även är vad som används i denna rapport för att undersöka just en av dessa algoritmer.

1.1 Syfte

Syftet med denna undersökning är att undersöka krypterings algoritmen AES, för att utveckla en förståelse för mer avancerade krypterings algoritmer. Samt att bygga en uppfattning om hur man på olika sätt kan implementera krypterings algoritmer och vad de får för betydelse för deras säkerhet och hastighet.

1.2 Frågeställningar

- Hur påverkas tiden de tar att kryptera något mellan de olika nyckel längderna 128-bit, 192-bit och 256-bit nyckel?
- Hur påverkas skifertexten av de olika körlägen och vilken betydelse får de för den resultatet?
- Hur förändras tiden det tar att kryptera något beroende på ifall algoritmen körs i ECB, CBC eller OFB samt vilken betydelse det får ur ett tillämpningsperspektiv?

1.3 Avgränsning

Denna rapport är en avgränsad undersökning av AES och dess användning som fokuserar på hur nyckellängd och körläge påverkar krypteringstiden. Detta samt hur den resulterande skiffr texten påverkas av ECB, CBC & OFB körlägena och hur detta i sin tur kan påverka säkerheten.

Denna analys av algoritmens säkerhet utelämnar faktorer så som möjliga attacker där ibland exempelvis Brute-Force² & Side-Channel³ attacker. Undersökningen är även begränsad till en mjukvaruimplementering och tar inte hänsyn till möjliga skillnader som kan uppstå när algoritmen implementeras på en hårdvarunivå.

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

²Neeraj 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.

³"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.⁴ 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 medelande i text då många i början inte kunde läsa till att idag istället använda komplexa algoritmer så som AES & DES.⁵ Begreppet kryptografi har dock också fått en utökade betydelse med tiden då det idag även inkluderar olika metoder för att säkerställa autenticiteten av informationen och avsändaren.⁶

2.1.1 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.⁷

Den tidiga kryptografin är även något som kan observeras hos romarna där man använde Caesarchiffer och hos grekerna. Där grekernas metod byggde på att man virade en pappersbit runt någon form av ett cylinderformat objekt och sedan skrev medellandet 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.⁸

2.1.2 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 medelande och hemlig kommunikation. Under den här tiden utvecklades bland annat Polyalphabetic substitutionsskiffer där ett av dom tidigaste skapades av Leon Battista Alberti.⁹

Där efter så försattes 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.¹⁰

Enigma kan man nästan se som ett av de första stegen i utvecklingen av den morderna kryptografin som till stora delar bygger på våran teknologiska utveckling. Den nya tekniken öppnade

⁴Wikipedia. *Kryptografi*. 2020. URL: <https://sv.wikipedia.org/w/index.php?title=Kryptografi&oldid=48532107> (hämtad 2022-09-07).

⁵Tony M Damico. "A brief history of cryptography". I: *Inquiries Journal* 1.11 (2009).

⁶Nationalencyklopedin. *kryptografi*. 2022. URL: <http://www.ne.se/uppslagsverk/encyklopedi/lng/kryptografi> (hämtad 2022-09-07).

⁷Dam09.

⁸Dam09.

⁹Dam09.

¹⁰Dam09.

nya portar, vilket bland annat gjorde det möjligt för krypteringen att bli mer komplicerad och säkrare utan att påverkar 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 medelande skickade av Tyskarna under andra världskriget och spelade på så sätt en ganska viktig roll i historien.¹¹

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 medelanden. Där ibland bland annat algoritmer som Advanced Encryption Standard (AES) och Data Encryption Standard (DES) men även protokoll som HTTP och SSH.¹²

2.2 AES Uppkomst

2.3 Tidigare forskning

¹¹Wikipedia. *Kryptografi*. 2020. URL: <https://sv.wikipedia.org/w/index.php?title=Kryptografi&oldid=48532107> (hämtad 2022-09-07).

¹²Wik20c.

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.¹³

För att den menade mottagaren ska kunna läsa skiftexten sedan så behöver hen ha en nyckel samt köra samma krypteringsalgoritm i dekrypterings läge. Bland de vanligaste typerna av krypteringsalgoritmer finns Symetrisk & Asymmetrisk Kryptering, vilket bland annat innehållar algoritmer som AES och RSA.¹⁴

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 bland annat.¹⁵

En ut av de 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.¹⁶

Blockskiffer används däremot inte bara för kryptering utan kan även användas i olika typer av Hashfunktioner och Pseudoslämp nummer generatorer bland annat då blockskifferets resulterande skiftext ser ut att vara slumpmässig.¹⁷

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 möjligt att istället kunna använda blockskiffer på större data mängder. 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.¹⁸

¹³Wikipedia. *Kryptering*. 2021. URL: <https://sv.wikipedia.org/w/index.php?title=Kryptering&oldid=49187134> (hämtad 2022-09-08).

¹⁴Wik21b.

¹⁵Wikipedia, the free encyclopedia. *Block cipher*. 2022. URL: https://en.wikipedia.org/w/index.php?title=Block_cipher&oldid=1111913955 (hämtad 2022-10-05).

¹⁶Wik22i.

¹⁷Wik22i.

¹⁸Wikipedia. *Block cipher mode of operation*. 2022. URL: https://en.wikipedia.org/w/index.php?title=Block_cipher_mode_of_operation&oldid=1106163325 (hämtad 2022-09-25).

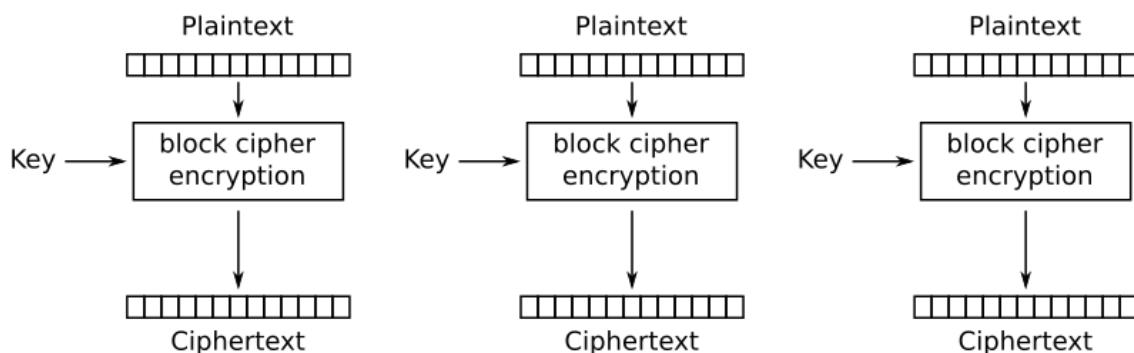
Ett ut av sätten som körlägen löser problemet med att applicera blockskiffer algoritmer på större data mängder är att dom använder sig av en så kallad Initialization Vector (IV). Det är en unik sekvens av bytes som används för att säkerställa att samma data mängd aldrig kommer att generera samma krypterade skifffertext.¹⁹

IV kan användas på många sätt från att introduceras genom en XOR-operation med första blocket och sedan kedja ihop och göra samma sak med nästkommande block fast med resultatet från den första operationen precis som i CBC. Detta gör att blocken blir beroende av varandra plus att samma information som krypteras flera gånger inte kommer ge samma resultat även fast man använder samma nyckel.²⁰

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 & 3.2.²¹

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.



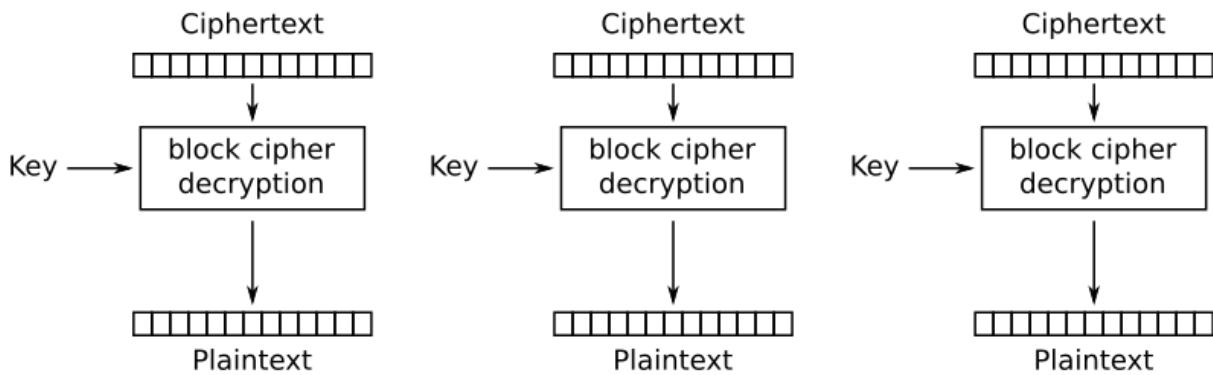
Figur 3.1: Electronic Code Book läge kryptering [[Wik22c](#)]

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

¹⁹[Wik22a](#).

²⁰[Wik22a](#).

²¹[Wik22a](#).



Figur 3.2: Electronic Code Book läge dekryptering [Wik22d]

På grund av ECB körlägets simplicitet 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.²²

Vad detta innebär är att för större mängder data är att det börjar bildas mönster i skiffertexten. 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 är ett säkert körläge och därför inte används näst intill aldrig i praktiken.²³

ECB har däremot även sina fördelar då de bland annat kan paralleliseras både när de gäller krypteringen och dekrypteringen. Detta samt att ECB även gör de möjligt att slumpmässigt dekryptera enskilda block av en skiffertext utan att man behöver dekryptera hela texten.²⁴

3.2.1.2 CBC

Cipher Block Chaining läge är ett av de 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 näckommande block innan de körs genom blockchiffer algoritmen.²⁵ Matematisk sett kan detta formuleras såhär:

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

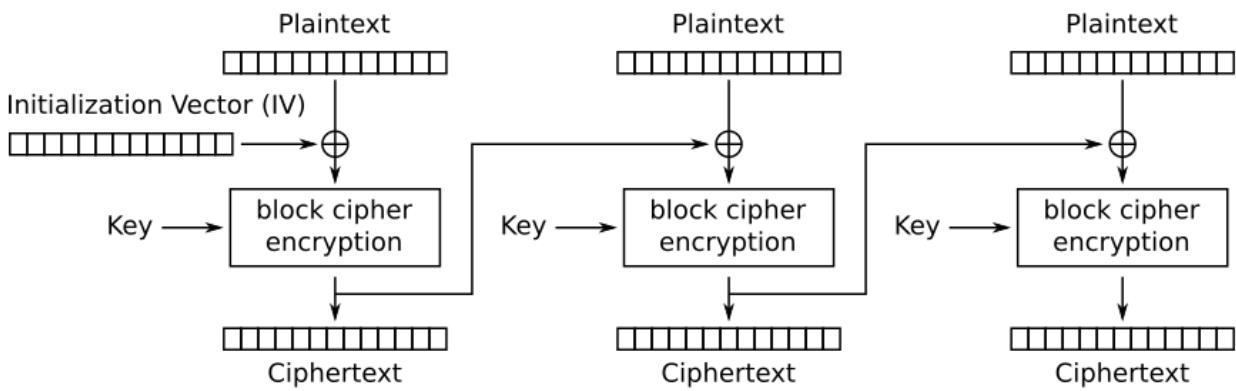
Där S_i är det krypterade blocket(skiffertexten), B_i är det blocket som ska krypteras, K_n är blockchiffer algoritmen där n står för nyckeln och S_{i-1} är det krypterade blocket före det blocket som ska krypteras. IV är en Initialization Vector (IV) som används vid krypteringen av de första blocket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

²²Wik22a.

²³Wik22a.

²⁴Wik22a.

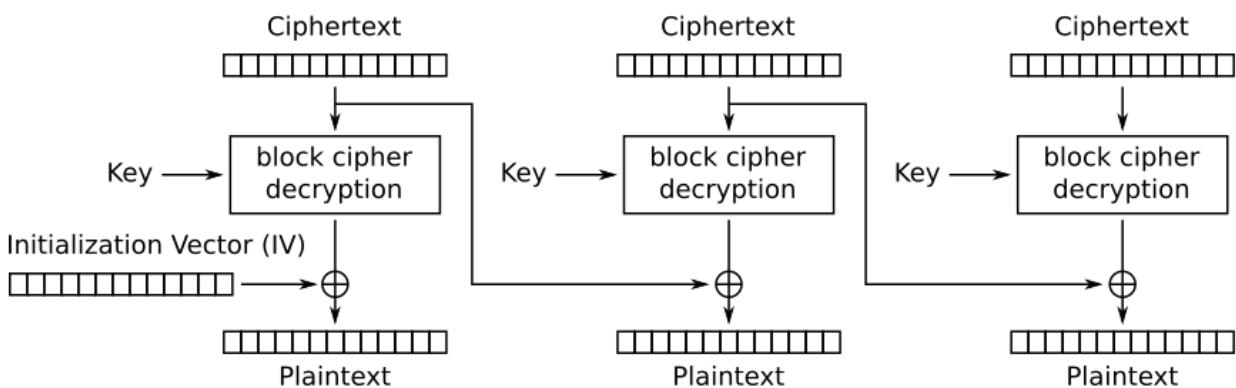
²⁵Wik22a.



Figur 3.3: Cipher Block Chaining läge kryptering [Wik22e]

När de gäller dekrypteringsprocessen för CBC så bär den precis som för ECB stora likheter med krypteringsprocessen. Det två skillnaderna som finns är att blockchiffert 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.²⁶ 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 [Wik22f]

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 medelanden kan ge samma krypterade block. Därför så krävs en Initialization Vector (IV) som används vid första blocket.²⁷

CBC är dock inte prefekt och har i sig också några nackdelar. Där ibland exempelvis de faktum att en incorrect IV leder till att de första blocket inte kan dekrypteras korrekt, detta påverkar dock inte de resterande blocken. På grund av det så kan man exempelvis lösa problemet genom

²⁶Wik22a.

²⁷Wik22a.

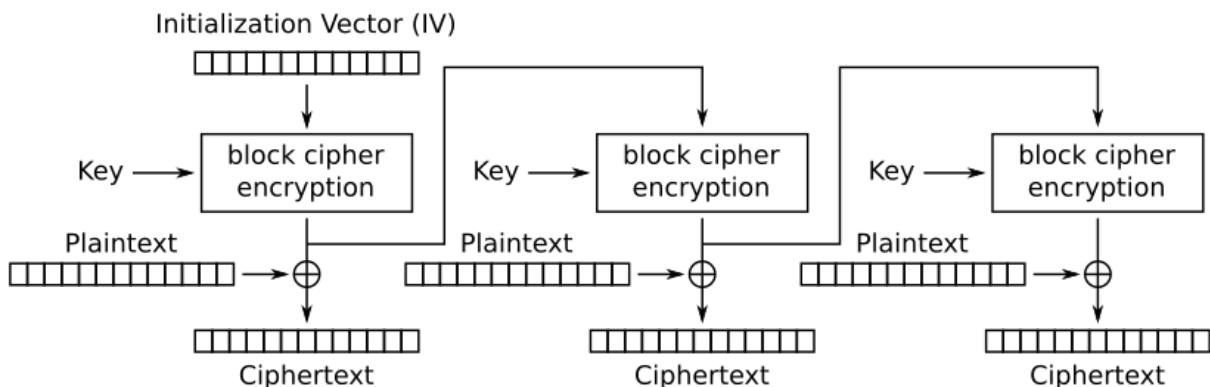
att första blocket bara innehåller någon typ av fyllnad, vilket då gör dekrypteringen möjlig utan tillgång till IV.²⁸

Utöver detta så begränsas även CBC till att bara vara parallelliserbar 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 skiffertexten.²⁹

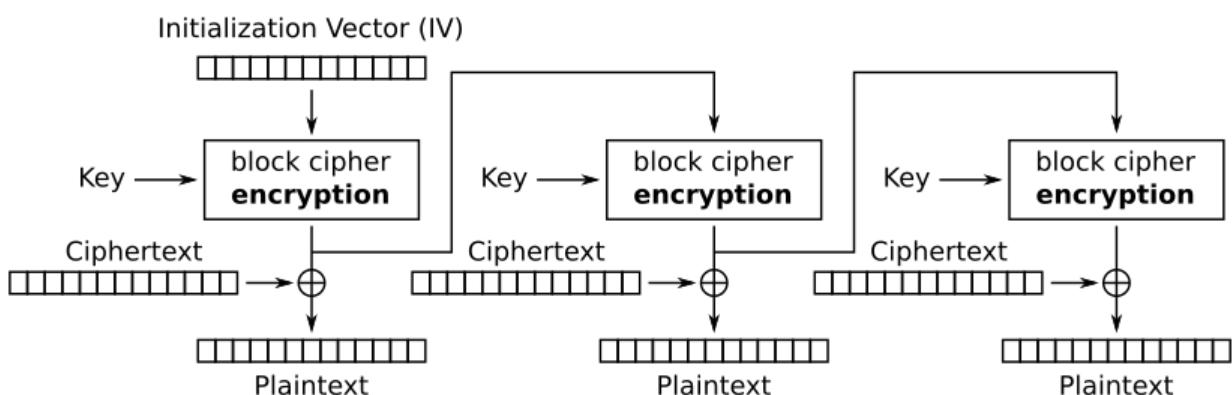
3.2.1.3 OFB

Output Feedback läge är ett ytterligare körläge som skiljer sig en del från ECB och CBC som redan presenterats. Den största skillnaden från det andra körlägena är att OFB inte använder blockchiffer algoritmen för att kryptera eller dekryptera blocken. Istället så körs IV genom blockchiffer algoritmen och den resulterande Nyckelström tillförs sedan genom en XOR-operation till blocket som ska krypteras eller dekrypteras.³⁰

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



Figur 3.5: Output Feedback läge kryptering [Wik22g]



Figur 3.6: Output Feedback läge dekryptering [Wik22h]

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

²⁸Wik22a.

Wik22a.

³⁰Wik22a.

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

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 Nyckelström som används för att kryptera eller dekryptera blocket. O_i i sin tur bygger då på att O_{i-1} körs genom blockchiffer algoritmen 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 blockskiffer operationerna i förväg, vilket gör det möjligt att även parallellisera OFB. Dock kan OFB inte parallelliseras ifall man inte gör blockskiffer operationerna i förväg. Utöver detta saknar även OFB möjligheten att slumpmässigt dekryptera enskilda block utan att behöva dekryptera hela skiffertexten.³¹

3.3 Symmetrisk & Asymmetrisk Kryptering

Symmetrisk och asymmetrisk kryptering handlar om hur nycklar används i olika krypteringsalgoritmer. För symmetriska krypterings algoritmer så betyder detta att samma nyckel är vad som används för både kryptering och dekryptering. Medans asymmetrisk kryptering bygger på att man använder olika nycklar för kryptering och dekrypterings processerna.³²

De symmetriska krypterings algoritmernas huvudsakliga nackdel ligger i de faktum att de krävs en delad känd nyckel mellan båda parter. Detta är något som asymmetriska krypterings algoritmer inte behöver, vilket har lett till att man ofta använder asymmetriska krypterings algoritmer för att sköta nyckelutbytet för de symmetriska krypterings algoritmerna. Anledningen till detta är att det symmetriska krypterings algoritmerna ofta är bättre för större data mängder då dom bland annat behöver mycket kortare nyckellängder.³³

Exempel på symmetriska krypterings algoritmer är bland annat AES och DES varav AES kommer förklaras djupare senare i denna rapport.³⁴ Medan exempel på asymmetriska krypterings algoritmer är bland annat RSA.³⁵

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 Blockskiffer krypterings algoritm som utformades för att ersätta DES som då var den dominerande krypterings algoritmen. AES är en av de mest använda krypterings algoritmerna idag och

³¹Wik22a.

³²Wikipedia. *Symmetric-key algorithm*. 2022. URL: https://en.wikipedia.org/w/index.php?title=Symmetric-key_algorithm&oldid=1106743629 (hämtad 2022-09-25).

³³Wik22b.

³⁴Wik22b.

³⁵Wikipedia, 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)³⁶. 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.³⁷

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 Nyckel utökning som alla kommer att förklaras mer detaljerat i de följande sektioner. Nyckel utökning delen kan även delas upp ytterligare i tre olika varianter som är Nyckel utökning 128bit, Nyckel utökning 192bit och Nyckel utökning 256bit beroende på vilken av de tre AES varianterna som används.³⁸

Utöver detta så bygger Nyckel utökning på tre operationer vilka är SubWord, RotWord och Rcon som också kommer att förklaras mer detaljerat i de följande sektioner.³⁹ För att förstå vissa av dessa delar så kommer det däremot krävas en förklaring av både AES S-Box och Finite Fields som kommer att förklaras först.

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.⁴⁰

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 en byte nivå 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.⁴¹ För en mer grundlig matematiskt förklaring av Finite Fields så kan man bland annat läsa rapporten *Ändliga kroppar* av Boman, Anna.⁴²

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 Nyckel utökning delen av AES algoritmen samt i SubBytes operationen.⁴³

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

³⁶Wikipedia, the free encyclopedia. *WPA*. 2021. URL: <https://sv.wikipedia.org/w/index.php?title=WPA&oldid=49187997> (hämtad 2022-10-26).

³⁷Wikipedia, the free encyclopedia. *Advanced Encryption Standard*. 2022. URL: https://en.wikipedia.org/w/index.php?title=Advanced_Encryption_Standard&oldid=1117488157#See_also (hämtad 2022-10-26).

³⁸Wik22a.

³⁹Wik22a.

⁴⁰Wikipedia, the free encyclopedia. *Finite field*. 2022. URL: https://en.wikipedia.org/w/index.php?title=Finite_field&oldid=1117661014 (hämtad 2022-10-27).

⁴¹Wik22k.

⁴²Boman, Anna. *Ändliga kroppar*. 2016.

⁴³Wikipedia, the free encyclopedia. *Rijndael S-box*. 2022. URL: 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 index värdet för att kunna hitta rätt värde i S-boxen.⁴⁴

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 attack vector 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.⁴⁵

AES har två olika S-boxar varav den ena är en invers S-box av den andra. Detta så att operationen går att utföra i båda riktningarna, vilket gör dekryptering av data möjlig.⁴⁶ Dessa två S-boxar går att se nedan i figur 3.7 där värdena är representerade i hexadecimala 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) [Wik22r]

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. Steg utförs antingen 10, 12 eller 14 gånger, vilket brukar kallas för rundor. 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.⁴⁷

AES algoritmen har en fixerad block storlek på 128bit vilket innebär att det alltid kommer att vara 16 Bytes som krypteras åt gången. Absolut först genomförs en initial AddRoundKey operationen operation där den första rund nyckeln används. Sedan genomförs 9, 11 eller 13 identiska rundor. Den sista rundan skiljer sig dock från det andra på så sätt att man

⁴⁴Wik22r.

⁴⁵Amandeep 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.

⁴⁶Wik22r.

⁴⁷Wik22a.

hoppar över sista MixColumns operationen då den inte får någon betydelse ur ett kryptografiskt perspektiv för den slutliga skiffrer texten.⁴⁸

En vanlig runda består av följande steg:

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

Den sista rundan ser istället ut på följande sätt:

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

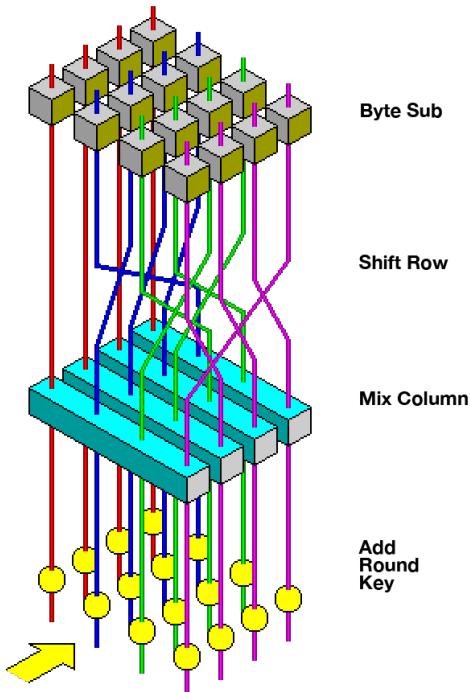
Utifrån detta kan man se att nyckeln används flera gånger under själva krypterings processen av ett block. För att varje runda då ska få en unik nyckel så genomförs en Nyckel utöknings operation som genererar en nyckel för varje runda utifrån den ursprungliga nyckeln. Detta för att samma nyckel inte ska användas flera gånger vilket skulle göra det enklare att bryta krypteringen.⁴⁹

Nyckel utökningen är en operationen som genomförs före själva krypterings stadiet påbörjas och genererar en lista med nycklar för varje runda. Antalet nycklar beror av nyckellängden som även bestämmer antalet rundor. För dekryptering så sker liknande rundor av operationer som för krypteringen fast i en lite annorlunda ordning samt med den inversa operationen till ShiftRows operationen & MixColumns operationen.⁵⁰

En vanlig dekrypterings runda ser då 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 invers operation handlar om hur dessa operationer är uppbyggda. För AddRoundKey operationen så är det på grund av XOR-operationens linjära egenskaper som innebär att den 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 de gäller SubBytes operationen så handlar de istället om att de helt enkelt är en substitution utifrån en AES S-Box, vilket



Figur 3.8: Vanlig runda [Wik99]

⁴⁸Wik22a.

⁴⁹Wik22a.

⁵⁰Wik22a.

då betyder att de som krävs för att genomföra den motsatta operationen blir att använda den inversa AES S-Boxen.⁵¹

Både krypterings och dekrypterings strukturen av ett block som beskrivits ovan kan representeras i Python kod på följande sätt i figur 3.9 & 3.10.

```
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
    for i in range(1, (nr - 1)):
        data = sub_bytes(data, subBytesTable)
        data = shift_rows(data)
        data = mix_columns(data)
        data = np.bitwise_xor(data, round_keys[i])

    # Final round
    data = sub_bytes(data, subBytesTable)
    data = shift_rows(data)
    data = np.bitwise_xor(data, round_keys[nr - 1])

return data
```

Figur 3.9: Krypterings runda funktion

```
def decryption_rounds(data, round_keys, nr):
    # Inizial add round key
    data = np.bitwise_xor(data, round_keys[-1])

    # Rounds 1 to 9 or 1 to 11 or 1 to 13
    for i in range(1, (nr - 1)):
        data = inv_shift_rows(data)
        data = sub_bytes(data, invSubBytesTable)
        data = np.bitwise_xor(data, round_keys[-(i+1)])
        data = inv_mix_columns(data)

    # Final round
    data = inv_shift_rows(data)
    data = sub_bytes(data, invSubBytesTable)
    data = np.bitwise_xor(data, round_keys[0])

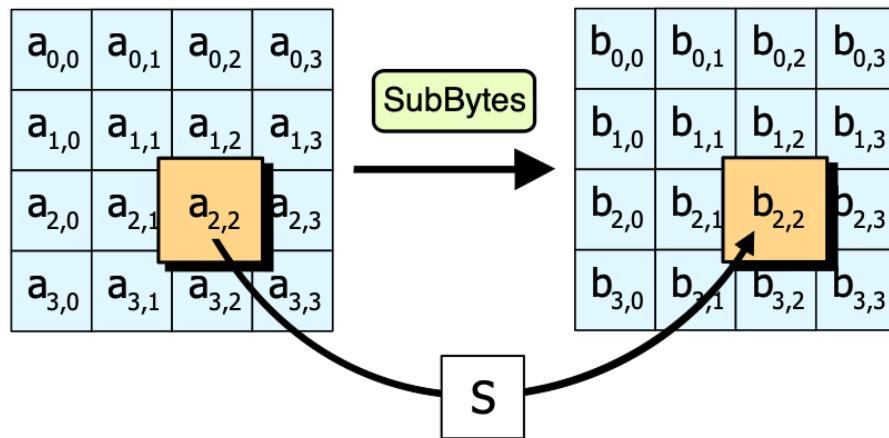
return data
```

Figur 3.10: Dekrypterings runda funktion

3.4.3.1 SubBytes operationen

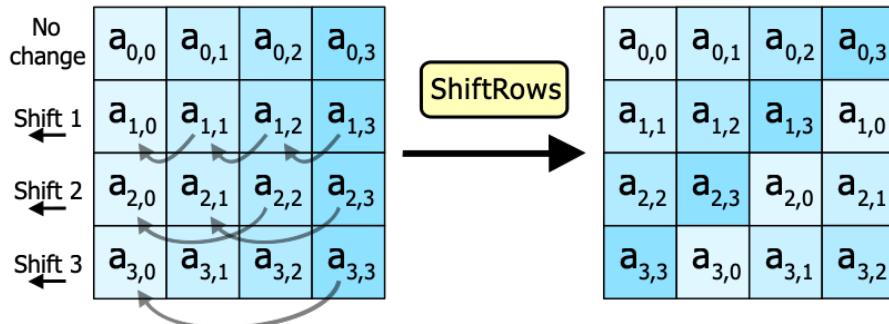
SubBytes operationen bygger på ...

⁵¹Wik22a.



Figur 3.11: SubBytes operationen [Wik21a]

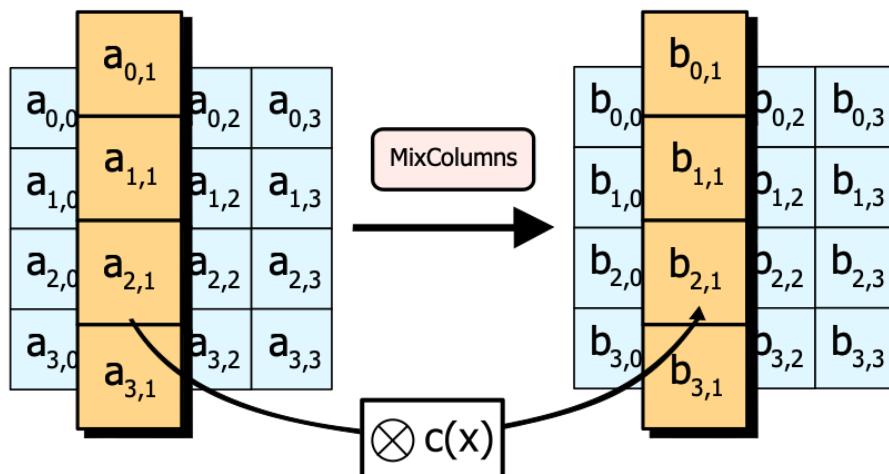
3.4.3.2 ShiftRows operationen



Figur 3.12: ShiftRows operationen [Wik20a]

3.4.3.3 Inverse ShiftRows operationen

3.4.3.4 MixColumns operationen



Figur 3.13: MixColumns operationen [Wik20b]

3.4.3.5 Inverse MixColumns operationen

3.4.3.6 AddRoundKey operationen

3.4.4 Nyckel utökning

3.4.4.1 RotWord

3.4.4.2 SubWord

3.4.4.3 Rcon

3.4.4.4 Nyckel utökning 128bit

3.4.4.5 Nyckel utökning 192bit

3.4.4.6 Nyckel utökning 256bit

3.4.5 AES-128bit

3.4.6 AES-192bit

3.4.7 AES-256bit

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. Själva 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 Python 3.11 som användes under undersöknings genomförandet.⁵²

4.1 Implementering

Implementeringen av AES är uppdelad i ett antal funktioner till stor del är baserat på hur strukturen och uppdelningen av AES som beskrivet i “AES proposal: Rijndael”⁵³. De Huvudsakliga funktionerna är följande:

- SubBytes operationen
- ShiftRows operationen
- MixColumns operationen
- AddRoundKey operationen
- Nyckel utökning

Dessa används då i själva algoritmens rundor samt utanför och är beskrivna och förklarade i respektive teori avsnitt. Utöver detta kan

4.2 Test Uppsättning

Test uppsättningen är uppdelad i tre delar där varje del är konstruerad för att generera ett resultat i koppling till frågeställningarna för denna undersökning.

4.2.1 Nyckellängds Test

4.2.2 Körläges Test

4.2.3 Krypterings Test

4.3 Genomförande

Genomförandet av undersökningen gick som följande:

⁵²Python Software Foundation. *Python 3.11.0*. 2022. URL: <https://www.python.org/downloads/release/python-3110/> (hämtad 2022-11-15).

⁵³DR99.

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. Det råa data värdena från hela undersökningen går att se i bilaga A, B.1 & B.2.

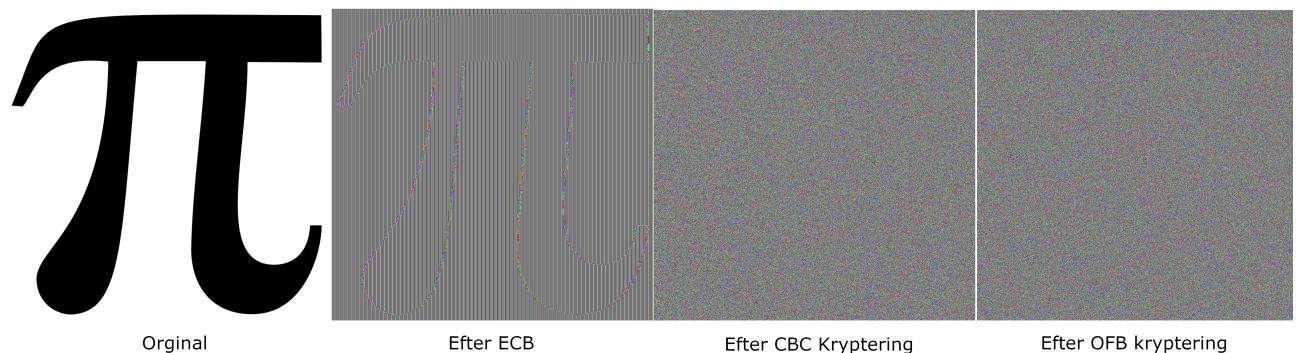
5.1 Nyckellängds Test

Resultat Tid (s)		
ECB - 128bit	ECB - 192bit	ECB - 256bit
21.05752382799983	25.014501572000153	29.177583716000083

5.2 Körläges Test

Resultat Tid (s)		
ECB - 128bit	CBC - 128bit	OFB - 128bit
20.82165230000013	20.885069635999972	20.89562146799988

5.3 Krypterings Test



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

6 Diskussion & Slutord

6.1 Felkällor

6.2 Förbättringar

6.3 Slutsats

6.4 Slutord

Källförteckning

- [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.
- [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.
- [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).
- [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.
- [Nat22] Nationalencyklopedin. *kryptografi*. 2022. URL: <http://www.ne.se/uppslagsverk/encyklopedi/lng/kryptografi> (hämtad 2022-09-07).
- [Pty22a] Python Software Foundation. *Python 3.11.0*. 2022. URL: <https://www.python.org/downloads/release/python-3110/> (hämtad 2022-11-15).
- [Pty22b] 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.
- [Wik20a] Wikipedia. *Advanced Encryption Standard*. 2020. URL: <https://commons.wikimedia.org/w/index.php?title=File:AES-ShiftRows.svg&oldid=482077415> (hämtad 2022-11-15).
- [Wik20b] Wikipedia. *Advanced Encryption Standard*. 2020. URL: <https://commons.wikimedia.org/w/index.php?title=File:AES-MixColumns.svg&oldid=488631026> (hämtad 2022-11-15).
- [Wik20c] Wikipedia. *Kryptografi*. 2020. URL: <https://sv.wikipedia.org/w/index.php?title=Kryptografi&oldid=48532107> (hämtad 2022-09-07).
- [Wik20] Wikipedia, the free encyclopedia. *Byte*. 2020. URL: <https://sv.wikipedia.org/w/index.php?title=Byte&oldid=48406212> (hämtad 2022-10-05).
- [Wik21a] Wikipedia, the free encyclopedia. *Binära talsystemet*. 2021. URL: https://sv.wikipedia.org/w/index.php?title=Binra_talsystemet&oldid=49765783 (hämtad 2022-10-17).
- [Wik21a] Wikipedia. *Advanced Encryption Standard*. 2021. URL: <https://commons.wikimedia.org/w/index.php?title=File:AES-SubBytes.svg&oldid=539923258> (hämtad 2022-11-15).
- [Wik21b] Wikipedia, the free encyclopedia. *Caesarchiffer*. 2021. URL: <https://sv.wikipedia.org/w/index.php?title=Caesarchiffer&oldid=48885737> (hämtad 2022-09-23).

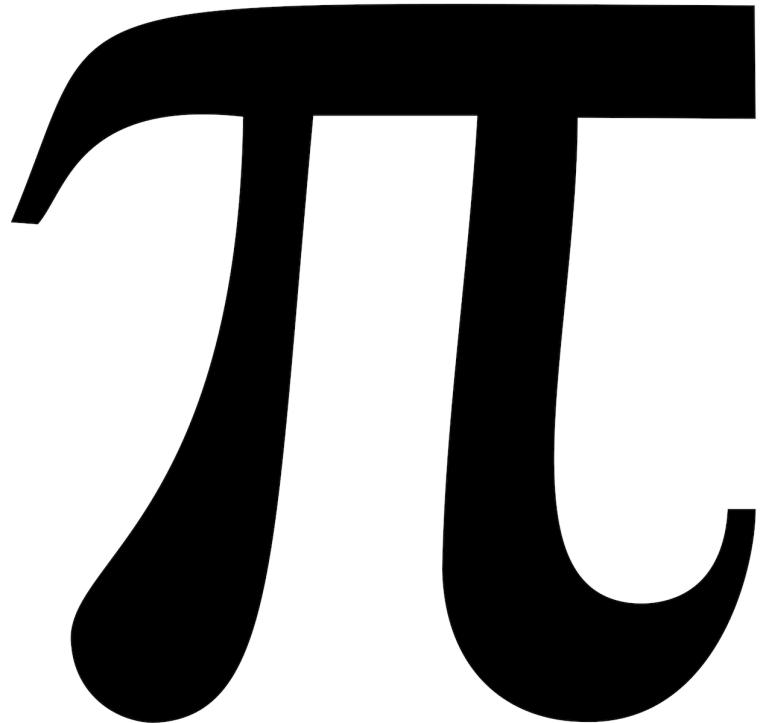
- [Wik21b] Wikipedia. *Kryptering*. 2021. URL: <https://sv.wikipedia.org/w/index.php?title=Kryptering&oldid=49187134> (hämtad 2022-09-08).
- [Wik21c] Wikipedia, the free encyclopedia. *Keystream*. 2021. URL: <https://en.wikipedia.org/w/index.php?title=Keystream&oldid=1039345792> (hämtad 2022-10-04).
- [Wik21d] Wikipedia, the free encyclopedia. *Visual Studio Code*. 2021. URL: https://sv.wikipedia.org/w/index.php?title=Visual_Studio_Code&oldid=48905230 (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).
- [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 (hämtad 2022-10-26).
- [Wik22a] Wikipedia. *Block cipher mode of operation*. 2022. URL: https://en.wikipedia.org/w/index.php?title=Block_cipher_mode_of_operation&oldid=1106163325 (hämtad 2022-09-25).
- [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 (hämtad 2022-09-25).
- [Wik22c] Wikipedia, the free encyclopedia. *Block cipher mode of operation*. 2022. URL: https://en.wikipedia.org/w/index.php?title=Block_cipher_mode_of_operation&oldid=1106163325#/media/File:ECB_encryption.svg (hämtad 2022-09-25).
- [Wik22d] Wikipedia, the free encyclopedia. *Block cipher mode of operation*. 2022. URL: https://en.wikipedia.org/w/index.php?title=Block_cipher_mode_of_operation&oldid=1106163325#/media/File:ECB_decryption.svg (hämtad 2022-09-25).
- [Wik22e] Wikipedia, the free encyclopedia. *Block cipher mode of operation*. 2022. URL: https://en.wikipedia.org/w/index.php?title=Block_cipher_mode_of_operation&oldid=1106163325#/media/File:CBC_encryption.svg (hämtad 2022-09-26).
- [Wik22f] Wikipedia, the free encyclopedia. *Block cipher mode of operation*. 2022. URL: https://en.wikipedia.org/w/index.php?title=Block_cipher_mode_of_operation&oldid=1106163325#/media/File:CBC_decryption.svg (hämtad 2022-09-26).
- [Wik22g] Wikipedia, the free encyclopedia. *Block cipher mode of operation*. 2022. URL: https://en.wikipedia.org/w/index.php?title=Block_cipher_mode_of_operation&oldid=1106163325#/media/File:OFB_encryption.svg (hämtad 2022-10-03).
- [Wik22h] Wikipedia, the free encyclopedia. *Block cipher mode of operation*. 2022. URL: https://en.wikipedia.org/w/index.php?title=Block_cipher_mode_of_operation&oldid=1106163325#/media/File:OFB_decryption.svg (hämtad 2022-10-03).
- [Wik22i] Wikipedia, the free encyclopedia. *Block cipher*. 2022. URL: https://en.wikipedia.org/w/index.php?title=Block_cipher&oldid=1111913955 (hämtad 2022-10-05).
- [Wik22j] Wikipedia, the free encyclopedia. *Enigma machine*. 2022. URL: https://en.wikipedia.org/w/index.php?title=Enigma_machine&oldid=1109006355 (hämtad 2022-10-17).
- [Wik22k] Wikipedia, the free encyclopedia. *Finite field*. 2022. URL: https://en.wikipedia.org/w/index.php?title=Finite_field&oldid=1117661014 (hämtad 2022-10-27).
- [Wik22l] Wikipedia, the free encyclopedia. *Frequency analysis*. 2022. URL: https://en.wikipedia.org/w/index.php?title=Frequency_analysis&oldid=1113626431 (hämtad 2022-10-10).

KÄLLFÖRTECKNING

- [Wik22m] Wikipedia, the free encyclopedia. *Hashfunktion*. 2022. URL: <https://sv.wikipedia.org/w/index.php?title=Hashfunktion&oldid=50669121> (hämtad 2022-10-07).
- [Wik22n] Wikipedia, the free encyclopedia. *Logical connective*. 2022. URL: https://en.wikipedia.org/w/index.php?title=Logical_connective&oldid=1113164184 (hämtad 2022-10-12).
- [Wik22o] Wikipedia, the free encyclopedia. *Polyalphabetic cipher*. 2022. URL: https://en.wikipedia.org/w/index.php?title=Polyalphabetic_cipher&oldid=1113775685 (hämtad 2022-10-10).
- [Wik22p] Wikipedia, the free encyclopedia. *Pseudorandomness*. 2022. URL: <https://en.wikipedia.org/w/index.php?title=Pseudorandomness&oldid=1112429322> (hämtad 2022-10-04).
- [Wik22q] Wikipedia, the free encyclopedia. *RSA*. 2022. URL: <https://sv.wikipedia.org/w/index.php?title=RSA&oldid=50280992> (hämtad 2022-10-04).
- [Wik22r] Wikipedia, the free encyclopedia. *Rijndael S-box*. 2022. URL: https://en.wikipedia.org/w/index.php?title=Rijndael_S-box&oldid=1110299033 (hämtad 2022-10-27).
- [Wik22s] Wikipedia, the free encyclopedia. *Stream cipher*. 2022. URL: https://en.wikipedia.org/w/index.php?title=Stream_cipher&oldid=1098861130 (hämtad 2022-10-05).
- [Wik22t] Wikipedia, the free encyclopedia. *Substitution cipher*. 2022. URL: https://en.wikipedia.org/w/index.php?title=Substitution_cipher&oldid=1111925704 (hämtad 2022-10-10).
- [Wik22u] Wikipedia, the free encyclopedia. *Substitution-permutation network*. 2022. URL: https://en.wikipedia.org/w/index.php?title=Substitution-permutation_network&oldid=1098155951 (hämtad 2022-11-10).
- [Wik99] Wikipedia, the free encyclopedia. *Advanced Encryption Standard*. 1999. URL: [https://en.wikipedia.org/wiki/Advanced_Encryption_Standard#/media/File:AES_\(Rijndael\)_Round_Function.png](https://en.wikipedia.org/wiki/Advanced_Encryption_Standard#/media/File:AES_(Rijndael)_Round_Function.png) (hämtad 2022-09-02).

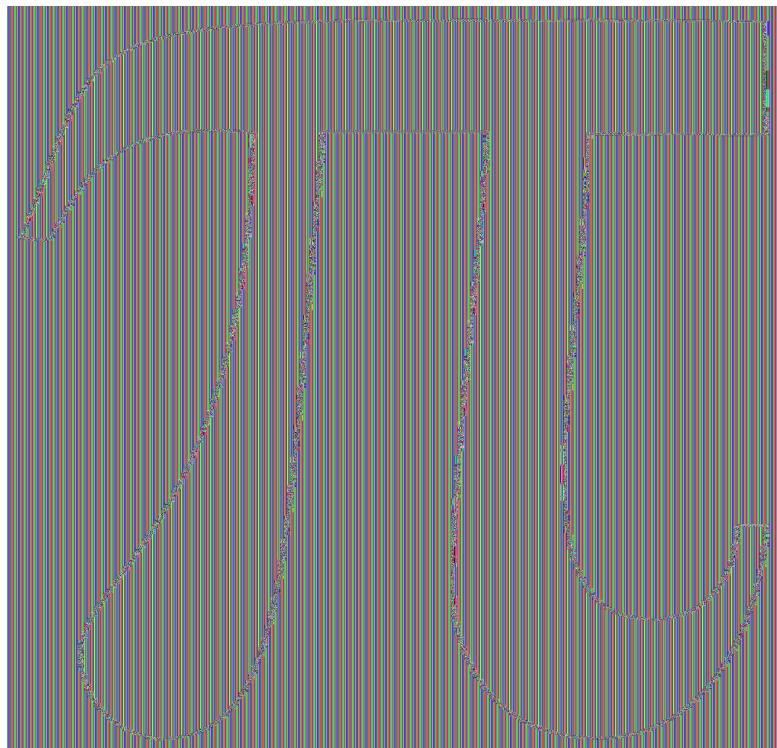
A AES körläges test resultat

A.1 Före testet



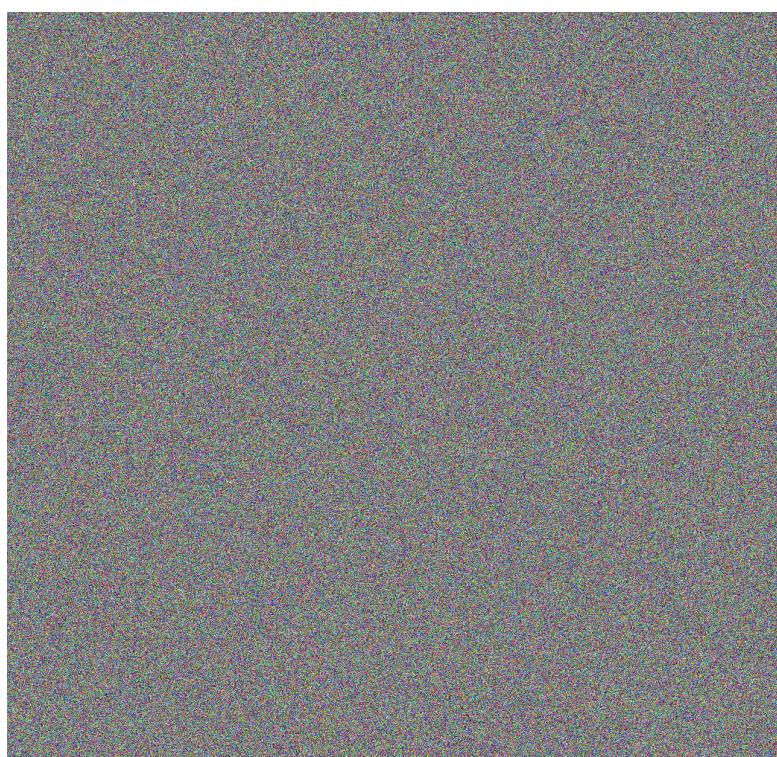
Figur A.1: Orginal bild

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 Rå data från test

B.1 Data från Nyckellängds test

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 körläges test

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
def xtime(a):
    return (((a << 1) ^ 0x1B) & 0xFF) if (a & 0x80) else (a << 1)

# Performs the byte substitution layer
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
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
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):
    def mix_single_column(data):
        # See Sec 4.1.2 in The Design of Rijndael
        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)

    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
def inv_mix_columns(data):
    # See Sec 4.1.3 in The Design of Rijndael
    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
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
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
    for i in range(1, (nr - 1)):
        data = sub_bytes(data, subBytesTable)
        data = shift_rows(data)
        data = mix_columns(data)
        data = np.bitwise_xor(data, round_keys[i])

    # Final round
    data = sub_bytes(data, subBytesTable)
    data = shift_rows(data)
    data = np.bitwise_xor(data, round_keys[nr - 1])

    return data

# Performs the decryption rounds
def decryption_rounds(data, round_keys, nr):
    # Inizial add round key
    data = np.bitwise_xor(data, round_keys[-1])

    # Rounds 1 to 9 or 1 to 11 or 1 to 13
    for i in range(1, (nr - 1)):
        data = inv_shift_rows(data)
        data = sub_bytes(data, invSubBytesTable)
        data = np.bitwise_xor(data, round_keys[-(i+1)])
        data = inv_mix_columns(data)

    # Final round
    data = inv_shift_rows(data)
    data = sub_bytes(data, invSubBytesTable)
    data = np.bitwise_xor(data, round_keys[0])

    return data

# -----
# Key expansion setup
# -----
# Key expansion function (returns a list of round keys)
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
    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

round_keys = [None for i in range(nr)]

tmp = [None for i in range(4)]

for i in range(nr * 4):
    for index, t in enumerate(words[i]):
        tmp[index] = int(t, 16) # type: ignore
    words[i] = tuple(tmp)

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

return round_keys, nr

# Key schedule (nk = number of columns, nr = number of rounds)
def key_schedule(key, nk, nr):
    # Create list and populates first nk words with 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 words, 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))
        words.append((xord[:2], xord[2:4], xord[4:6], xord[6:8]))
    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 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 value from sbox based on the current word
def SubWord(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 without prefix
        piece = hex(subBytesTable[sBoxIndex])[2:]

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

        sWord.append(piece)

    # Return 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
        raw, 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)
    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

```
from PyAES.encrypt import encrypt
from PyAES.decrypt import decrypt
from getpass import getpass
import PyAES

def main():
    print("-"*85)
    print(r"""
    / \   | - - | . ' - - \   | - - \   / | - [ | --. .--.
    / ___ \   | _| - - . - - ' . | | - - | | - - / [ \ [ ] | | | .-. | / . ' \ 
    -/ /   \ \_ -| | - / | | \ - - ) | | - - | | - - \ ' / / | | , | | | | | \ - - .
    | - - | | - - | | - - | \ - - . ' | | - - | | - - | \ : / \ - - / [ - - ] | - - ] ' . - - ' 
    """)
```

```

        \_._.'



print("-"*85)
print(f"Version: {PyAES.__version__}

                                            {PyAES.__copyright__}")

print("-"*85)
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)

```
from PyAES.encrypt import encrypt
from time import perf_counter
from random import randint
from os import remove

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

def write_data(data, name_f):
    with open(name_f + '.txt', 'w') as f:
        for i in data:
            f.write(str(i) + '\n')
        f.write(str('\n'))

def setup(count):
    with open("test_speed.txt", 'wb') as f:
        for j in range(count):
            f.write(bytes([randint(0, 255)]))

def speed_test(count, key, mode, iv):
    setup(count)
    start = perf_counter()
    encrypt(key, "test_speed.txt", mode, iv=iv)
    end = perf_counter()
    remove("test_speed.txt.enc")
    return end - start

if __name__ == '__main__':
    # Run in parallel
    keys = ["2b7e151628aed2a6abf7158809cf4f3c",
            "8e73b0f7da0e6452c810f32b809079e562f8ead2522c6b7b",
            ""
            "603deb1015ca71be2b73aef0857d77811f38"]
    iv = "000102030405060708090a0b0c0d0e0f"
    file_size = 1000000
    runs = 25

    # Test time difference between 128, 192 and 256 bit keys
    data = []
    progress = 0
    for i in keys:
```

```

data_tmp = []
for j in range(runs):
    progress = progress_bar(progress, 150)
    test = speed_test(file_size, i, 'ECB', iv)
    data_tmp.append(test)
write_data(data_tmp, 'keys_test_raw' + str(len(i) * 4))
data.append(sum(data_tmp)/len(data_tmp))
write_data(data, 'keys_test')

# Test time difference between ECB, CBC and OFB modes
data = []
data_tmp = []
for i in range(runs):
    progress = progress_bar(progress, 150)
    test = speed_test(file_size, keys[0], 'OFB', iv)
    data_tmp.append(test)
write_data(data_tmp, 'modes_test_raw OFB')
data.append(sum(data_tmp)/len(data_tmp))
data_tmp = []
for i in range(runs):
    progress = progress_bar(progress, 150)
    test = speed_test(file_size, keys[0], 'CBC', iv)
    data_tmp.append(test)
write_data(data_tmp, 'modes_test_raw CBC')
data.append(sum(data_tmp)/len(data_tmp))
data_tmp = []
for i in range(runs):
    progress = progress_bar(progress, 150)
    test = speed_test(file_size, keys[0], 'ECB', iv)
    data_tmp.append(test)
write_data(data_tmp, 'modes_test_raw ECB')
data.append(sum(data_tmp)/len(data_tmp))
write_data(data, 'modes_test')

progress = progress_bar(progress, 150)
print("\n")
print("Completed")

```