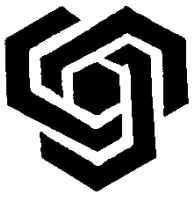


ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – СОФИЯ

ФАКУЛТЕТ КОМПЮТЪРНИ СИСТЕМИ И ТЕХНОЛОГИИ

КУРСОВА РАБОТА

по

Криптографски методи за защита на информация

Тема: Симетрична криптография

Изготвил:

Боян Зарев
факултетен номер: 123222004
група: 42
e-mail: bzarev@tu-sofia.bg
Курс III
факултет и специалност: ФКСТ, КСИ

Ръководител:

ас. Михаела Асенова

13.03.2025 г.

Резюме

Курсовата работа разглежда основни принципи и класификация на криптографските алгоритми, като се фокусира върху симетричното криптиране. В началото е представен общ преглед на криптирането и неговото значение за информационната сигурност. Разгледани са поточните и блоковите шифри, като са описани техните предимства и недостатъци.

При поточните шифри е анализиран Linear-feedback shift register (LFSR) – ключов компонент, използван за генериране на псевдослучайни последователности. Блоковите шифри са разгледани в контекста на техните режими на работа, които влияят върху сигурността и ефективността на криптирането.

Подробно е описана мрежата на Файстел, която е основа за много съвременни криптографски алгоритми. След това е представен AES (Advanced Encryption Standard), включително неговата структура, основни операции и методът за разширяване на ключа (Key Expansion).

Основен принос на курсовата работа е предложената модификация на AES-128, с която се изследва как промяната влияе върху лавинния ефект. Чрез експерименти е направено сравнение с оригиналния AES-128, като са анализирани получените резултати.

Работата завършва с обобщение на направените изводи и преглед на използваната литература, която обосновава теоретичните и практически аспекти на изследването.

Съдържание

Увод

Живеем в система, в която обмена на информацията се осъществява чрез интернет по бърз и ефективен начин. Но обаче тази информация може да бъде използвана от хора, които имат недоброжелателни намерения. Поради тази причина, необходимо е тази информация да бъде защитена. Под сигурна информация се подразбира, да до нея имат достъп малък и подбран брой хора, като това се осигурява чрез криптиране. С развитието на технологията, нуждата за по-силни криптиращи алгоритми расте. Сред методите за защита на данни криптирането е най-доброто решение за информационна сигурност, тъй като е способно да защитава различни видове данни. Сигурността може да бъде дефинирана като механизъм, който прави невъзможен всеки неоторизиран или нежелан достъп до информация и услуги. [1]

Криптирането е техника за преобразуване на нормален четим текст в нечетима форма с използването на алгоритъм и ключ. Ключът е поредица от байтове, с която определения алгоритъм разбърква съдържанието на съобщението. След криптирането корелацията между съдържанието на съобщението и шифровия текст става не толкова очевидна. Този текст може да се разшифрова само с помощта на ключа, който първоначално се е използвал за разбъркване на информацията.

1. Класификация на криптиранията

Криптиранията могат да бъдат класифицирани в три основни типа въз основа на използвания алгоритъм и начина, по който се използва съответния ключ:

- *Симетрично криптиране*

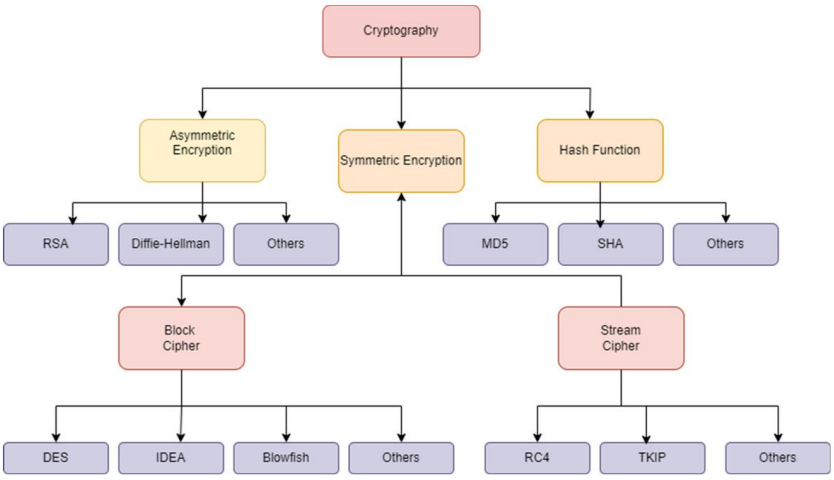
  При този метод един и същ ключ се използва както за криптиране, така и за декриптиране на данните. Методът е бърз и ефективен, но изисква сигурен обмен на ключовете между изпращащият и получателя. Лесни са за реализация, както софтуерно така и хардуерно. Да кажем, че в има мрежа има n потребителя, то тогава за всеки потребител ще са необходими n(n-1)/2 ключа общо. Симетричните алгоритми се делят на: поточни и блокови. Те са подробно описани по-надолу.

- *Асиметрично криптиране*

  Този метод използва два различни ключа – публичен ключ за криптиране и частен ключ за декриптиране. Методът е по-надежден от симетричните криптиранията, но за сметка на това много е по-бавен. Тези криптирания се използват при електронни подписи.

- *Хеширане*

  Хеширането е еднопосочен процес, при който данните се преобразуват в поредица от байтове с фиксирана дължина с помощта на хеш функция. Хеш функция е функция, която преобразува оригиналния текст без използване на ключ. Процесът е необратим.



(фиг. 1) Класификация на криптиранията

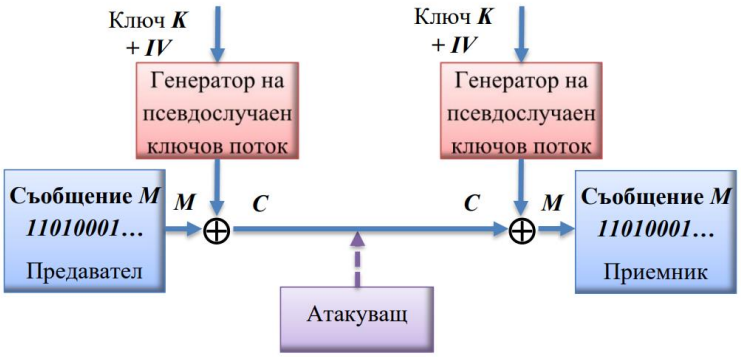
2. Видове симетрични криптирания

Както е наведено по-горе, симетричното криптиране е един от основните видове криптографски алгоритми, при който един и същ ключ се използва както за криптиране, така и за декриптиране на информацията. Това означава, че изпращачът и получателят трябва да притежават един и същ ключ, за да могат да обменят сигурно данни.

Симетричните алгоритми обикновено са по-бързи в сравнение на асиметричните, тъй като работят с по-кратки ключове и по-опростени математически операции. Поради високата си скорост, симетричното криптиране се използва за защита на големи количества информация, например база данни и мрежови комуникации.

2.1. Поточни шифри (stream ciphers)

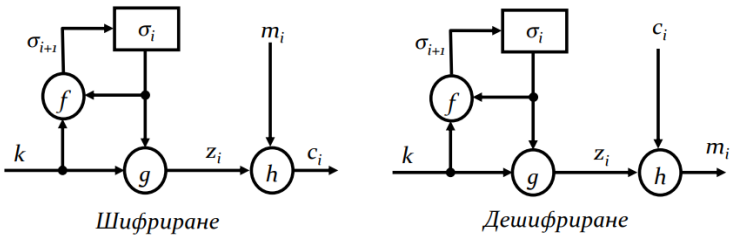
Поточният шифър е симетричен шифър, който обработва съобщението като поток от битове като изпълнява математически функции с всеки бит в даден момент от време. Поточните шифри използват генератори на ключов поток, които произвеждат псевдослучаен поток от битове, който се наслагва с открития текст чрез операцията сума по модул 2.



(фиг. 2) Схема на работа на поточен шифър

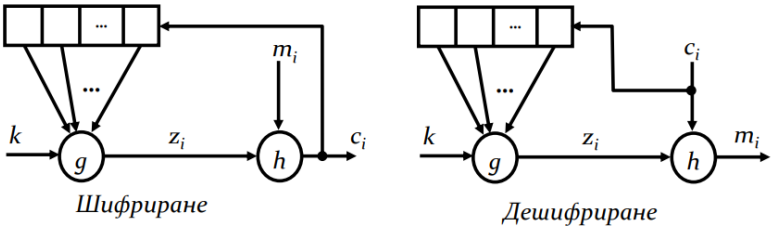
Векторите за инициализация IV (Initialization Vector) са произволни стойности, които се използват със симетричните алгоритми, за да се гарантира, че по време на процеса на криптиране не са създадени повтарящи се специфични последователности. Използват се заедно с ключовете K и не трябва да бъдат криптирани, когато се изпращат до приемника. IV и K заедно осигуряват случайността на процеса на криптиране.[2]

Синхронен поточен шифър е шифър, при който ключовия поток се генерира независимо от потока данни и потока на шифротекста.



(фиг. 3) Шифриране и дешифриране на синхронен поточен шифър

Асинхронен или самосинхронизираш се шифър е такъв шифър, при който ключовата последователност се генерира като функция на ключа и на фиксиран брой предходни състояния на разрядите на шифротекста.



(фиг. 4) Шифриране и дешифриране на асинхронен поточен шифър

Характеристика	Синхронни поточни шифри	Асинхронни поточни шифри
Синхронизиране	Необходимост от ресинхронизиране при грешка.	Самосинхронизиране след *t* такта.
Разпространение на грешки	Без разпространение на грешки, грешка само в сгрешения бит.	Ограничено разпространение на грешките в *t* бита след сгрешения бит.
Активни атаки	Устойчиви на активни атаки.	Неустойчиви на активни атаки.
Статистически свойства	Липса на разсейване на статистическото разпределение на открития текст.	Разсейване на статистическото разпределение на открития текст.

(таб. 1) Характеристики на синхронни и асинхронни поточни криптирания

2.1.1. Linear-feedback shift register

Linear-feedback shift register (LFSR) е регистър, чийто вход е линейна функция от предишни негови състояния. Принцип на неговата работа е изместването на старите битове и изчислението на новия бит, който след изчисленията се слага в началото на поредицата от битове. Бинарната операция, която се използва при изчисление на новия бит е сума по модул 2 (XOR). Те се използват при реализация на псевдогенератори на числа, които се използвам при поточните шифри.

Добър LFSR е този, който генерира максимално дълъг период от псевдослучайни битове. Ако поредицата се състои от m бита, то тогава максималният период на поредицата е $2^m - 1$. Максималният период не е 2^m защото поредицата, където всички битове са нули, винаги ще дава 0 на изхода, понеже функцията, с която изчисляваме новия бит е сума по модул 2 и тази поредица не я броим.

LFSR да бъде псевдошумов трябва да отговаря на трите постулата на Голомб:

1. Броят на единиците на изхода във всеки цикъл от периода на редицата да се не различава от броя на нулите на изхода с не повече от 1.
2. Във всеки цикъл половината от сериите с еднакви битове да имат дължина 1, четвърт от тях да имат дължина 2, осма от тях да имат дължина 3 и т.н.
3. Ако вземем цялата псевдослучайната последователност и я изместим циклично с слагане на първия бит в края ще се получи нова последователност, която изглежда като началната, но с различна начална точка. Описано математически, дадената автокорелационна функция долу е необходимо да има само 2 нива. Изместването се осъществява с left shift (първия бит става последен).

$$C(\tau) = \frac{1}{m} \sum_{k=0}^{n} a_n \oplus a_{n+\tau}$$

m – броя на битовете в поредицата

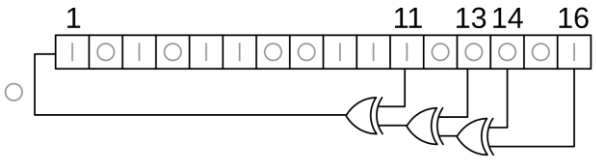
τ – изместване на поредицата [3]

Полиномът на обратните връзки определя схемата на LFSR. Първия и последния бит винаги участват във функцията ($q_L \neq 0$, $q_0 \neq 0$).

$$q(x) = q_L x^L + q_{L-1} x^{L-1} + \ldots + q_1 x - 1$$

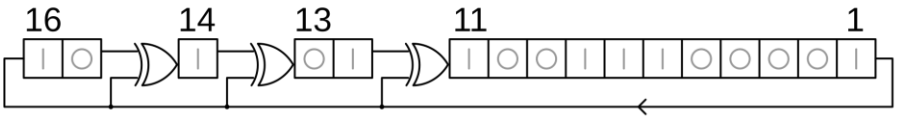
Съществуват 2 архитектури за реализация на LFSR:

- LFSR на Фибоначи – битовете се изместват надясно и новият бит се получава от сума по модул 2 от всичките определени предишни битове. Тази архитектура е лесна за реализация, но е сравнително бавна, защото обратната връзка влияе върху всичките битове.



(фиг. 5) 16-bit LFSR на Фибоначи

- LFSR на Галоа – всеки бит се измества и изчислява локално, без да се променят всички битове наведнъж. Обратната връзка се добавя директно към конкретните битове, а не само към първия бит. Битовете се придвижват от дясно на ляво, но за разлика Фибоначи номерацията на битовите клетки е в обратен редослед (n, n-1, n-2, …, 1).



(фиг. 6) 16-bit LFSR на Галоа

2.2. Блокови шифри

В съвременните системи за блоково шифриране сложните операции на разместване на изходните блокове се осъществява чрез поредица от прости заместявания и разместявания с цел опростяване на извършваните операции. Възлите реализиращи тези преобразувания се наричат съответно P-кутии (Permutation box) S-кутии (Substitution box).[2]

Идеята на блоковите шифри е промяната на един бит в съобщението да не съответства само на промяната на един бит в шифровия текст, както при поточните шифри, което помага за по-лесното пробиване на алгоритъма. Целта е да криптираната информация зависи от повече бита в съобщението. С други думи казано, ако един бит от входното съобщение се промени, трябва да се промени голяма част от шифротекста. Това свойство се нарича Diffusion (разсейване). За тази цел се използват P-кутии.

Още едно важно свойство на алгоритъма е Confusion (объркване). Това свойство е връзката между шифротекста и ключа. Тя трябва да е сложна и нелинейна. За тази цел се използват S-кутиите.

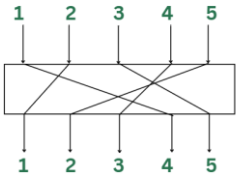
Блоковите шифри обработват данните на части с фиксиран размер, наречени блокове, вместо да шифрират бит по бит, както е при поточните шифри.

P-кутиите служат за разпространение на статистическите характеристики на входното съобщение върху шифротекста. Някои съобщения притежават статистически шаблони, които могат да бъдат запазени дори след криптиране. Например, ако най-честата буква в съобщението е „а“, тя ще бъде по-склонна да претърпи най-чести промени на съответните места при криптиране, което може да улесни анализа и дешифрирането. P-кутиите се използват, за да се намали този ефект и да се затрудни разкриването на тези шаблони, като осигуряват равномерно разпределение на информацията върху шифротекста.[4]

Видове P-кутии:

- Straight P-box

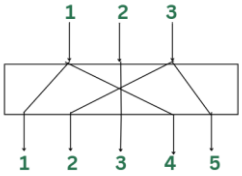
  При този тип броят на входовете и изходите е еднакъв. На изходите всъщност се намират всички входовете, но в разбъркан редослед.



(фиг. 7) Straight P-box

- Expansion P-box

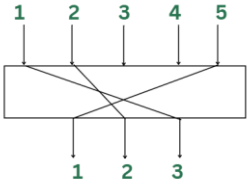
  Този тип има n входа и m изхода, където n<m. Броят на изходите е по-голям от броя на входовете, което означава, че един вход може да съответства на повече изхода.



(фиг. 8) Expansion P-box

- Compression P-box

  Този тип P-кутия има n входа и m изхода, където n>m. Броят на входовете е по-голям от броя на изходите, което означава, че в процеса на шифриране някои битове от информацията ще се „изтрият“, т.е. няма да се запишат в шифротекста.



(фиг. 9) Compression P-box

Обратимостта на P-кутиите е свойство, което гарантира, че при дешифрирането може да бъде възстановена първоначалната последователност от битове. Това означава, че ако се подаде на дешифриращия алгоритъм изхода на P-кутиите, получен по време на шифрирането, ще се възстанови входа им в оригиналния му вид. Това свойство имат само Straight P-кутиите, защото при тях разбъркването е проследимо. Но в другите два типа това не е случай. При Expansion P-кутиите не може да се знае със сигурност каква е оригиналната поредица, защото повече входа съответстват на един изход. При Compression P-кутиите някои битове се изтрити и при декриптирането е невъзможно да се възстановят.

S-кутията е основен елемент в много блокови шифри, използван за замяна на входни битови последователности с други битови последователности по предварително зададена таблица. Основната ѝ цел е да въведе нелинейност в криптографския алгоритъм. Операцията, която тя извършва е субституция на входните битове. Обикновено S-кутията приема m входни бита и ги преобразува в n изходни бита, като n не е задължително равно на m. S-кутия с размер m×n може да бъде реализирана като таблица за търсене (lookup table) с 2^m думи, всяка съдържаща n-битов изход. Обикновено се използват фиксирани таблици, както в алгоритъма DES. В някои шифри обаче таблиците се генерират динамично от ключа, например в алгоритмите Blowfish и Twofish.[5]

Пр. Таблица на S-кутия от алгоритъма DES (S_5), която прави субституция на 6-битов вход в 4-битов изход.

S_5		Middle 4 bits of input															
		0000	**0001**	**0010**	**0011**	**0100**	**0101**	**0110**	**0111**	**1000**	**1001**	**1010**	**1011**	**1100**	**1101**	**1110**	**1111**
Outer bits	**00**	0010	1100	0100	0001	0111	1010	1011	0110	1000	0101	0011	1111	1101	0000	1110	1001
	01	1110	1011	0010	1100	0100	0111	1101	0001	0101	0000	1111	1010	0011	1001	1000	0110
	10	0100	0010	0001	1011	1010	1101	0111	1000	1111	1001	1100	0101	0110	0011	0000	1110
	11	1011	1000	1100	0111	0001	1110	0010	1101	0110	1111	0000	1001	1010	0100	0101	0011

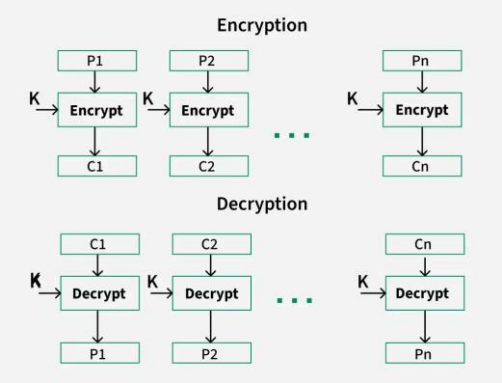
(таб. 2) S_5 от DES

Нека входът бъде 011011_2. Двубитовото число, което правят първия и последния бит е 01_2. Това число е реда на съответния изход. Останалата част от първоначалното число 1101_2 е колоната на съответния изход. Следователно при вход 011011_2 таблицата S_5 дава 1001_2 на изхода.

2.2.1. Режими на използване на блокови шифри

1. Електронна книга (ECB – Electronic Code Book)

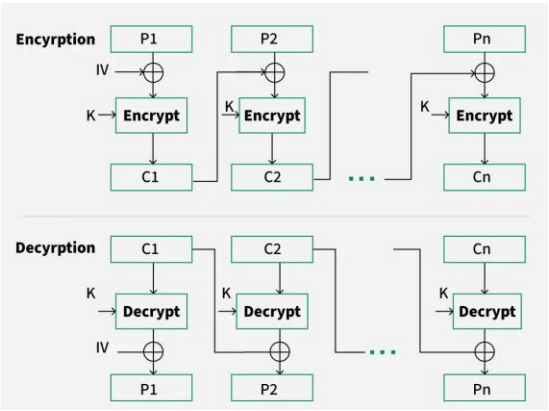
Електронният кодов буквар (ECB) е най-простият режим на работа за блокови шифри. Той се отличава с лесната си реализация, като всеки блок от входния текст се шифрира независимо, а изходът е в същата форма — блокове от шифрован текст. Ако съобщението е по-голямо от b бита, то се разделя на съответния брой блокове и за всеки от тях се прилага същата процедура.



(фиг. 10) ECB схема

2. Свързване на блокове във верига (CBC – Cipher Block Changing)

CBC е напредък спрямо режима ECB, тъй като ECB не осигурява достатъчно висока сигурност. При CBC, последният шифров блок се използва като вход за следващото шифриране, след като бъде подложен на XOR операция с текущия блок от входния текст. В основата на този процес стои генерирането на нов шифров блок чрез шифриране на резултата от XOR операцията между предишния шифров блок и настоящия блок от входния текст.

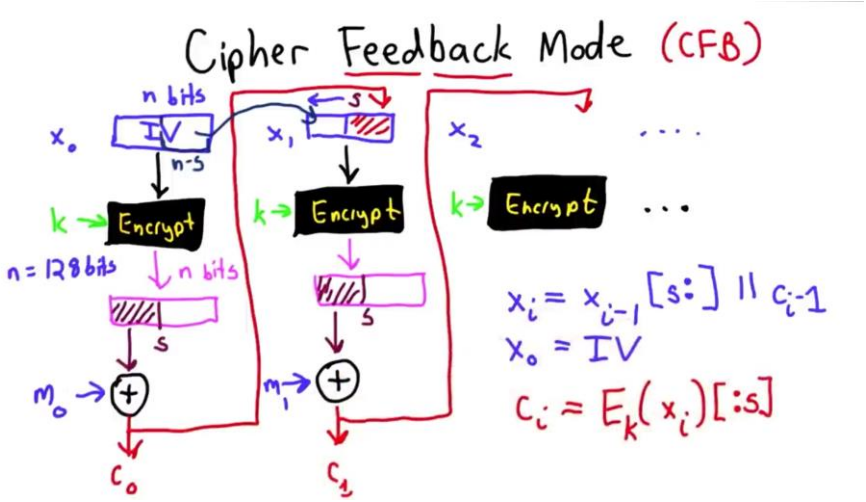


(фиг. 11) CBC схема

3. Режим на обратна връзка по шифър (CFB – Cipher Feed Back)

В този режим шифърът се използва като обратна връзка за следващия блок на шифриране, като се прилагат нови спецификации: първоначално се използва начален инициализираш вектор (IV) за първоначалното шифриране, а изходните битове се разделят на два набора. Ако блоковете са с дължина b, то тогава изходните блокове ще се разделят на s и b-s бита, където s е произволно число по-малко от b. Лявата страна на блока, съдържаща s бита, се избира заедно с битовете на входния текст, върху които се прилага операция XOR. Резултатът на това е първия криптиран блок.

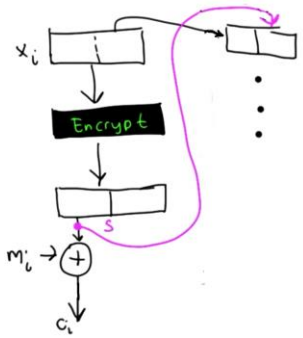
Следващата стъпка е подаване на входния параметър за криптиращата функция към регистъра за изместване. Този входен параметър в началото е инициализиращия вектор, но след това в следващите итерации е съдържанието на предишния изместваш регистър. Регистъра за изместване прави операцията left shift с s места. s празните бита се попълват с последния криптиран блок.



(фиг. 12) CFB схема

4. Режим на обратна връзка по изход (OFB – Output Feed Back)

При този режим подходът е същият както при CFB, но вместо сайфър текста към изместващия регистър се подава лявата част на изхода на криптиращата функция (s бита от блока след функцията Encrypt). Останалата част (b-s бита) се взима от предишния изместващ регистър, или ако е в началото от инициализиращия вектор. За разлика от CFB, ако един сайфър блок се загуби, другата част от криптираното съобщение може да се възстанови, защото то не зависи от миналите поредици от сайфър текста.



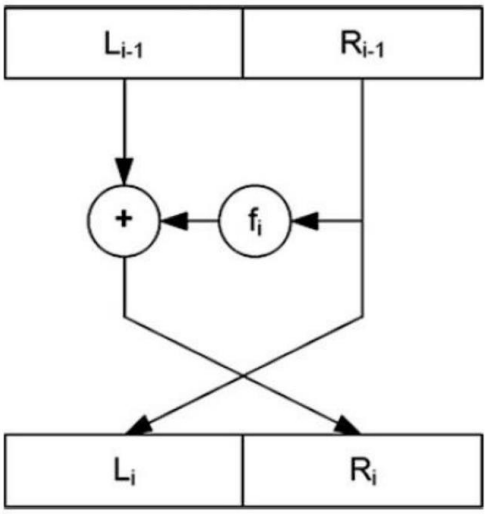
(фиг. 13) OFB схема

3. Мрежа на Файстел

Feistel мрежата е симетрична структура, използвана в криптографията за проектиране на блокови шифри, при която входните данни се разделят на две половини и се обработват през множество рундове с помощта на рундова функция и ключ.

Първоначално блока се разделя на лява и дясна част. Дясната част се преобразува с някаква нелинейна функция и ключ и резултатът от нея се сумира по модул 2 с лявата част. Резултатът от тази операция се записва отдясно. Дясната част се пренася без промяна в лявата част на резултатния блок. Операцията може да се повтори многократно. На последната стъпка крайният резултат е в обърнат вид без да включва в себе си нелинейна функция.[6]

При декриптиране, криптирането съобщение се слага в началото на мрежата и на изхода се получава първоначалното съобщение. Обаче реда на подаването на ключовете към функцията е обратен в отношение при процеса на криптирането.



(фиг. 14) Мрежа на Файстел

4. AES (Advanced encryption standard)

AES е стандарт за криптиране, приет от Националния институт и технологии на САЩ (NIST) през 2001 г. Той е направен като алтернатива на по-стария алгоритъм DES (Data Encryption Standard) и е широко използван за защита на данни. AES е блоков симетричен алгоритъм. Размерът на блока е 128 бита, докато размера на ключа може да бъде от 128, 192 и 256 бита. Стандартът използва алгоритъма Rijndael разработен от Joan Daemen и Vincent Rijmen.

Процедурата на криптиране и декриптирането с този алгоритъм включва функцията на разширяване на ключа (Key Expansion). Тази функция взима ключа като вход и като изход дава по един ключ за всяка рунда на алгоритъма. Броя на рундовете зависи от дължината на ключа. За 128 битов ключ са необходими 10 рунда, за 192 бита - 12, за 256 - 14 рунда.

Всеки рунд без последния се състоя от 4 стъпки:

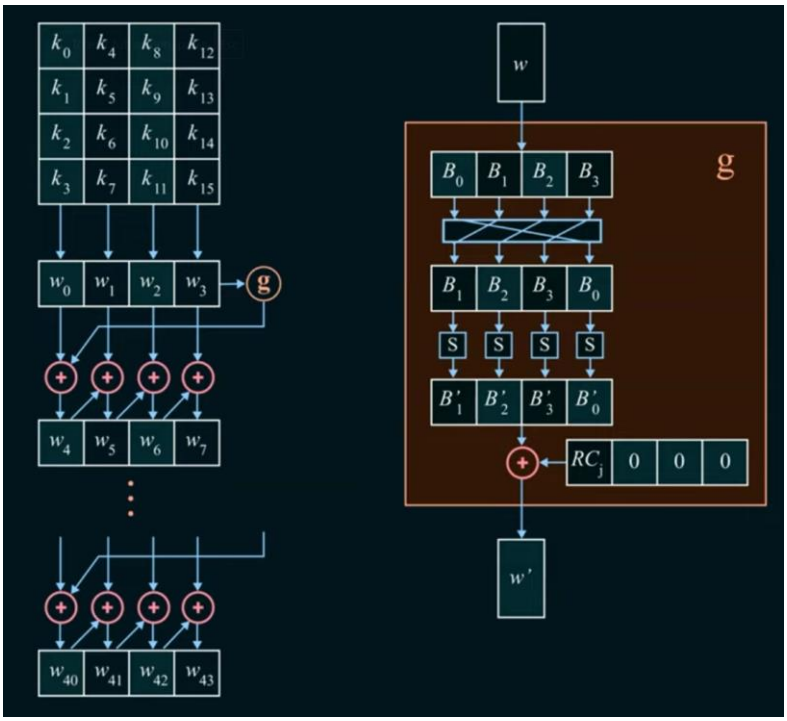
1. SubBytes
2. ShiftRows
3. MixColumns
4. AddRoundKey

Функцията Key Expansion разделя целия ключ на думи от 32 бита. Ако ключа е от 128 бита, то тогава функцията ще работи с 4 думи. Думите от предишния етап преминават през XOR и през функция g, преобразувайки се в думи, които ще съставят новия ключ за следващата рунда.

Във функцията g определената дума преминава първо през P кутия. След това разместените байтове се преобразуват със S кутиите на AES. Накрая полученият байт минава през сума по модул 2 със байтовете RC_j 00 00 00, където RC_j за първите 10 рунда са:

i	1	2	3	4	5	6	7	8	9	10
rc_i	01	02	04	08	10	20	40	80	1B	36

(таб. 3) RC_j при първите 10 рунда



(фиг. 15) Схема на AES Key Expansion

1. SubBytes

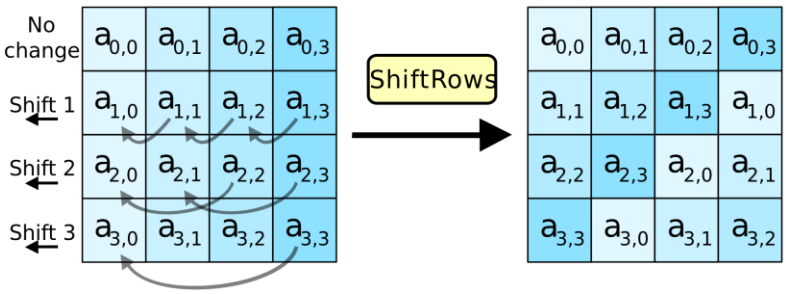
Всеки байт се занемя с друг байт, използвайки S-box таблица. Това заместване се прави така, че един байт никога да не се замести със себе си или със своята инверсия. Крайният резултат от етапа/рунда е отново таблица 4 x 4 (16 байта). В етапа SubBytes всеки байт в State се заменя със стойността от фиксирана 8-битова таблица, S; $b_{ij} = S(a_{ij})$. В етапа се пресмята мултипликативния обратен елемент на полинома от полето $GF(2^8)$. Използва се неразложимия полином $m(x) = x^8 + x^4 + x^3 + x + 1$. Първите 4 бита от байта са реда в таблицата на новия байт, а вторите 4 - колоната. [7]

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

(таб. 4) S-box на AES

2. ShiftRows

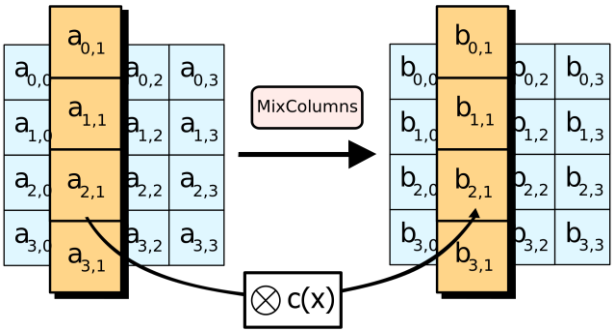
В този етап байтовете във всеки ред от масива State се изместват циклично наляво. Първия ред остава същия. Във втория клетките се местят за едно място вляво, във третия за 2, в четвъртия за 3 места.



(фиг. 16) Визуализация на стъпката ShiftRows

3. MixColumns

Етапът MixColumns по своята същност представлява умножение на матрици. Всяка колона на масива State се умножава със специфична матрица и така всеки байт от колоната се променя в резултата. Тази стъпка се пропуска в последния рунд (етап) от алгоритъма.[8]



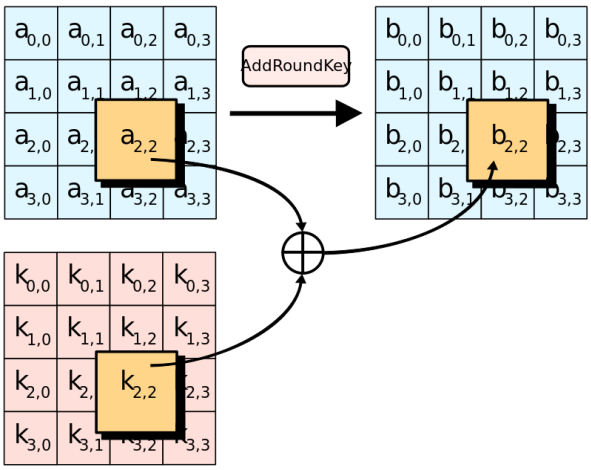
(фиг. 17) Визуализация на стъпката MixColumns

$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix}$$

(фиг. 18) Матрицата в стъпката MixColumns

4. AddRoundKey

В тази стъпка всеки байт от блока на основното съобщение минава през сума по модул 2 със всеки байт от съответстващият round ключ от функцията Key Expansion.



(фиг. 19) Визуализация на стъпката AddRoundKey

5. Модификация на AES-128. Сравнение на лавинния ефект с оригиналния AES-128

Лавинният ефект (avalanche effect) е фундаментално свойство на криптографските алгоритми, особено в симетричните шифри като AES. Той се отнася до това, че дори малка промяна в входните данни (например промяна на един бит в ключа или в откритият текст) води до значителна и непредвидима промяна в шифрования текст. Добър шифър при промяна на 1 бит променя 50% от битовете в поредицата.

Модификацията на AES-128 включва промяна на функцията Key Expansion. Направени са тестове, които показват сравнение на лавинния ефект при стандартния AES-128 ECB и модифицирания AES-128 използвайки NIST тестовите вектори.

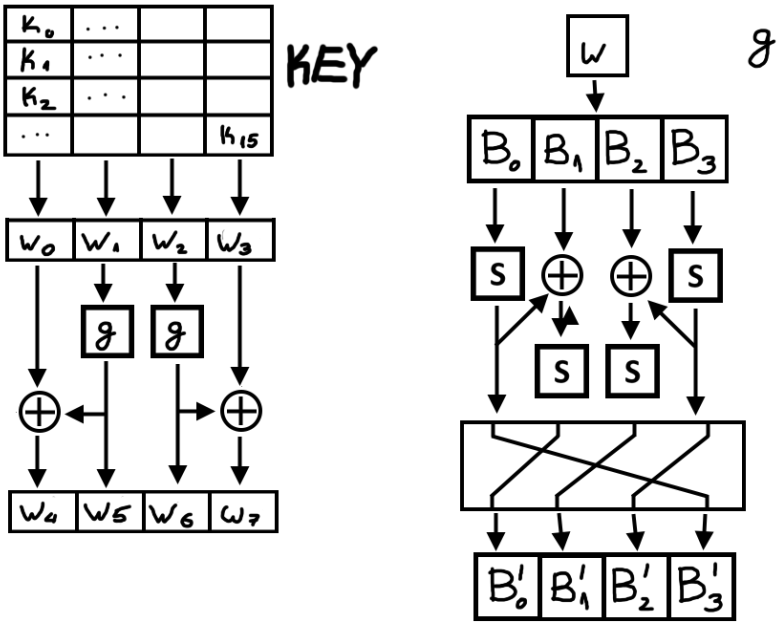
Тестовите вектори представляват низове, където всеки char е число от шестнайсетична бройна система. По същество това число е 4 бита от байт от ключа или от основния текст.

F.1.1 ECB-AES128.Encrypt

Key	2b7e151628aed2a6abf7158809cf4f3c
Block #1	
Plaintext	6bc1bee22e409f96e93d7e117393172a
Input Block	6bc1bee22e409f96e93d7e117393172a
Output Block	3ad77bb40d7a3660a89ecaf32466ef97
Ciphertext	3ad77bb40d7a3660a89ecaf32466ef97
Block #2	
Plaintext	ae2d8a571e03ac9c9eb76fac45af8e51
Input Block	ae2d8a571e03ac9c9eb76fac45af8e51
Output Block	f5d3d58503b9699de785895a96fdbaaf
Ciphertext	f5d3d58503b9699de785895a96fdbaaf
Block #3	
Plaintext	30c81c46a35ce411e5fbc1191a0a52ef
Input Block	30c81c46a35ce411e5fbc1191a0a52ef
Output Block	43b1cd7f598ece23881b00e3ed030688
Ciphertext	43b1cd7f598ece23881b00e3ed030688
Block #4	
Plaintext	f69f2445df4f9b17ad2b417be66c3710
Input Block	f69f2445df4f9b17ad2b417be66c3710
Output Block	7b0c785e27e8ad3f8223207104725dd4
Ciphertext	7b0c785e27e8ad3f8223207104725dd4

(фиг. 20) Тестови вектори на NIST за AES-128 ECB

Модификацията на функцията на разширяване на ключа включва g функция с P и S кутии, които са по стандарт на AES. Първия и четвъртия байт се преобразуват с S кутии, след което получените байтове се сумират по модул две с втория и третия байт – $S_1' \oplus S_2$, $S_4' \oplus S_3$ за да се получат новите битове за втория и третия байт на думата. Втората и третата дума преминават през тази функция, след което със сума по модул 2 със първата и втората дума дават думите за следващата рунда.



(фиг. 20) Схема на модифицираната Key Expansion функция

$$N - sizeOfWord$$
$$s - substitutionFunction$$
$$p - permutationFunction$$
$$k = N * j, j = \{4 * x | x \epsilon N\}$$
$$w'_k = s(w_k)$$
$$w'_{k+3} = s(w_{k+3})$$
$$w'_{k+1} = s(w'_k \oplus w_{k+1})$$
$$w'_{k+2} = s(w_{k+2} \oplus w'_{k+3})$$
$$w_{k+N}, w_{k+N+1}, w_{k+N+2}, w_{k+N+3} = p(w'_k, w'_{k+1}, w'_{k+2}, w'_{k+3})$$

```python
# MODIFIED PART OF AES
def g(word):
    # word is 32bits or 4 bytes
    byte1, byte2, byte3, byte4 = devide_bytes(word)

    next_byte_1 = substitution(byte1)
    next_byte_4 = substitution(byte4)

    next_byte_2 = next_byte_1 ^ byte2
    next_byte_3 = byte3 ^ next_byte_4

    next_byte_2 = substitution(next_byte_2)
    next_byte_3 = substitution(next_byte_3)

    output = (next_byte_1 << 24) | (next_byte_2 << 16) | (next_byte_3 << 8) | next_byte_4
    output = p_box(output)
    return output


def key_expansion(key_raw_bytes):
    words = [int.from_bytes(key_raw_bytes[i:i+4], byteorder="big") for i in range(0, len(key_raw_bytes), len(key_raw_bytes) // 4)]

    keys_output = [] # array of ints
    key_row = []

    for i in range(11): # 1 key per round, there is 10 rounds + 1 key for initialializing block
        # round
        words[1] = g(words[1])
        words[2] = g(words[2])

        words[0] ^= words[1]
        words[3] ^= words[2]

        for j in range(4):
            key_row.extend(devide_bytes(words[j]))
        keys_output.append(key_row)
        key_row = []
        # round
    #for i in range(11): # visualizing key expansion
    #    print(list(map(hex, keys_output[i])))
    return keys_output
# # #
```

```python
= [
  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

def substitution(byte):
    row = (byte >> 4) & 0xF
    column = byte & 0xF
    return s[row * 16 + column]
```

```python
def devide_bytes(word):
    byte1 = (word >> 24) & 0xFF
    byte2 = (word >> 16) & 0xFF
    byte3 = (word >> 8) & 0xFF
    byte4 = word & 0xFF

    return (byte1, byte2, byte3, byte4)

def p_box(word):
    byte1, byte2, byte3, byte4 = devide_bytes(word)

    output = (byte2 << 24) | (byte3 << 16) | (byte4 << 8) | byte1
    return output
```

(фиг. 21) Код от модифицираната Key Expansion функция

Сравнението на лавинния ефект направих върху четирите тестови вектора посочени във фиг. 20. При всеки вектор е сменян всеки бит и броени са променените битове при стандартния AES и модифицирания. Формулата, по която се изчислява ефекта е:

$$\text{ЛавиненЕфект} = \frac{\text{Брой на променени битове в } C}{\text{Общ брой битове в блока}} \times 100\%$$

Получените проценти се осредняват за да се получи крайния отговор, описващ лавинния ефект при двата алгоритъма.

```
Key:                          2b7e151628aed2a6abf7158809cf4f3c

Test 1:
Plaintext:                    6bc1bee22e409f96e93d7e117393172a
AES128:                       3ad77bb40d7a3660a89ecaf32466ef97
AES modified KeyExpnasion:    5e14be2842fda518e965c35e7dc72a9c
Avalanche effect
Original: 49.87%              Modified: 12.54%

Test 2:
Plaintext:                    ae2d8a571e03ac9c9eb76fac45af8e51
AES128:                       f5d3d58503b9699de785895a96fdbaaf
AES modified KeyExpnasion:    47e44a1da97a6ab533cd45d186306eaf
Avalanche effect
Original: 50.80%              Modified: 12.54%

Test 3:
Plaintext:                    30c81c46a35ce411e5fbc1191a0a52ef
AES128:                       43b1cd7f598ece23881b00e3ed030688
AES modified KeyExpnasion:    ac50316dd4bc9cd1ff2dd779f4c86ebd
Avalanche effect
Original: 50.60%              Modified: 12.10%

Test 4:
Plaintext:                    f69f2445df4f9b17ad2b417be66c3710
AES128:                       7b0c785e27e8ad3f8223207104725dd4
AES modified KeyExpnasion:    3b3f316a8c26eaee47eefce14090cec3
Avalanche effect
Original: 50.30%              Modified: 12.23%
```

(фиг. 22) Резултати от тестовете за лавинния ефект

Заключение:

Стандартният AES има по-добри резултати при лавинния ефект, защото има по-добра функция за разширяване на ключа. Лавинния ефект при оригиналния алгоритъм е около 50%, а при модифицирания е около 12%.

Кодът е качен в репоситори на следния линк: https://github.com/boce1/KMZI_AES128_keyexpansion_modification.git

Литература

[1] A Comparative Analysis of AES, RSA, and 3DES Encryption Standards based on Speed and Performance (Mirza Hamza Munir Baig1, Hafiz Burhan Ul Haq1, * , Waleed Habib1)

[2] Лекция 3 – Поточни шифри, КМЗИ, слайд 10 (доц. д-р Антония Ташева, проф. д.н. Жанета Савова)

[3] Neso Academy - Golomb's Randomness Postulates (youtube, 4:37 timestamp) https://www.youtube.com/watch?v=6agTBs3c89o

[4] Spanning Tree - AES: How to Design Secure Encryption (youtube, 8:16 timestamp) https://www.youtube.com/watch?v=C4ATDMIz5wc&t=69s

[5] Wikipedia – S-box, https://en.wikipedia.org/wiki/S-box

[6] Лекция 4 – Симетрични криптирания (Част 1), КМЗИ, слайд 8 (доц. д-р Антония Ташева, проф. д.н. Жанета Савова)

[7] Лекция 5 – Симетрични криптирания (Част 2), КМЗИ, слайд 18 (доц. д-р Антония Ташева, проф. д.н. Жанета Савова)

[8] Лекция 5 – Симетрични криптирания (Част 2), КМЗИ, слайд 22 (доц. д-р Антония Ташева, проф. д.н. Жанета Савова)

AES demo - https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38a.pdf

Computerphile – Feistel cipher, https://www.youtube.com/watch?v=FGhj3CGxl8I

Computerphile – S and P boxes, https://www.youtube.com/watch?v=DLjzI5dX8jc

Computerphile – AES, https://www.youtube.com/watch?v=O4xNJsjtN6E&t=29s

Wikipedia – Shift регистър, https://en.wikipedia.org/wiki/Linear-feedback_shift_register

Geek for Geeks – P box, https://www.geeksforgeeks.org/what-is-p-box-in-cryptography/

Udacity – CFB, https://www.youtube.com/watch?v=ASR8TDIf_6c

Udacity – OFB, https://www.youtube.com/watch?v=GsSJt6NZF9E

Modified AES Algorithm Using Multiple S-Boxes (Felicisimo V. Wenceslao, Jr., Bobby D. Gerardo, Bartolome T. Tanguilig III)