KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
Multimedijos inžinerijos katedra

MARIUS DIDJURGIS

DUOMENŲ APSAUGOS METODŲ TYRIMAS

Magistro baigiamasis darbas

Vadovas:
doc. Armantas Ostreika

KAUNAS, 2010

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
MULTIMEDIJOS INŽINERIJOS KATEDRA

TVIRTINU
Katedros vedėjas
doc. D. Rubliauskas
2010-06-01

DUOMENŲ APSAUGOS METODŲ TYRIMAS

Informatikos magistro baigiamasis darbas

Kalbos konsultantė
Lietuviu kalbos katedros lektorė
dr. J. Mikelionienė
2010-05-23

Recenzentas
doc. E. Karčiauskas
2010-05-30

Vadovas
doc. A. Ostreika
2010-05-30

Atliko
IFM 8/1 gr. stud.
M.Didjurgis
2010-05-30

KAUNAS, 2010

KVALIFIKACINĖ KOMISIJA

Pirmininkas:	Laimutis Telksnys, akademikas
Sekretorius:	Stasys Maciulevičius, docentas
Nariai:	Rimantas Barauskas, profesorius
	Raimundas Jasinevičius, profesorius
	Jonas Kazimieras Matickas, docentas
	Jonas Mockus, akademikas
	Rimantas Plėštys, docentas
	Henrikas Pranevičius, profesorius

Summary

Data protection problem is relevant in now days living. This paper takes a review of the existing methods and programs in digital data cryptography. Also it explains AES (advanced encryption standard) method, the most trustful one and overviews software tool, that was coded and used in experiments analysing this algorithm.

The new AES class, that was created in this project, is compared using few parameters with the older one, created by Microsoft. Despite the only purpose, to create a better program, the older implemented AES cipher class and it's methods beats the new one.

TURINYS**Error! Bookmark not defined.**

1. ĮVADAS

Šių dienų pasaulis pilnas kompiuterinės technikos įvairovių. Be jų mes neįsivaizduojame savo gyvenimo. Esant kompiuterinei pažangai, žmogus daug savo veiksnių perkėlė į skaitmeninę tikrovę, tai dokumentai, parašai, knygos, straipsniai, asmens tapatybės duomenys ir panašiai. Tokius ir panašius dalykus laikydami skaitmeniniame pavidale mes palengviname gyvenimą sau, kitiems, o taip pat ir gamtai. Žmogus sutaupo laiko, jėgų ir išteklių. Nebereikia nešiotis su savimi pinigų, dokumentų, knygų. Palengvėja gyvenimas perkant ar parduodant internetu prekes bei paslaugas, bendraujant su žmonėmis, perduodant įgaliojimus ar kokius tai dokumentus.

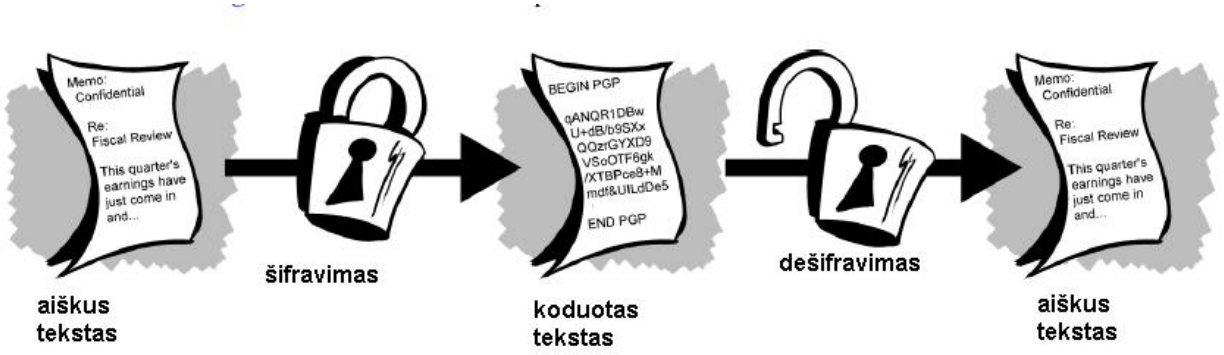
Tačiau negalima teigti, kad žmogaus pinigai yra visiškai saugūs, nes jų nebereikia nešiotis su savimi, negali sakyti, kad niekas nepasinaudos tavo duomenimis ar privatumu. Tobulėjant technologijoms, vagys taip pat prisitaiko prie kintančios aplinkos. Tokiame technikos amžiuje vis daugėja elektroninių vagysčių. Yra sukuriama įvairių šnipinėjimo programų, kompiuterinių virusų, „trojos“ arklių, kurios vagia skaitmeniniame pavidale esančią informaciją. Pavogus žmogaus tapatybės duomenis ar kreditinės kortelės numerius vagis gali prisipirkti prekių internetu, pavogus slaptus dokumentus, jais galima įvairiai manipuliuoti ar šantažuoti. Todėl iškyla ta pati problema, kaip apsisaugoti nuo nesankcionuoto priėjimo prie slaptų dokumentų, kaip apsaugoti savo duomenis nuo trečiųjų šalių.

Šio darbo tikslas yra padaryti kriptografijos apžvalgą, peržiūrėti kokie metodai ir jų algoritmai egzistuoja šiomis dienomis, vieną jų realizuoti programiniame lygmenyje ir ištirti pagal keletą parametrų.

2. KRIPTOGRAFIJOS APŽVALGA

Kriptografija – rašymo būdas įslaptinti teksto turinį.

Žmonės nuo senos raštuose, piešiniuose arba schemose įamžina savo patirtį, atradimus. Kiekvienas gali juos perskaityti ir suprasti ir naudoti kitų patirtį savo reikmėms. Tokia, visiems suprantama kalba išreikšta informacija kriptografijoje vadinama aiškiais duomenimis ar informacija. Informacijos užmaskavimas ir padarymas ją neperskaitomą vadinamas šifravimu, o tokie duomenys – koduotais duomenimis. Informacijos užšifravimas naudojamas tam, kad jos negalėtų perskaityti ir ja pasinaudoti, tie asmenys, kuri jiems nėra skirta. Procesas, kuriais koduoti duomenys padaromi vėl suprantami ir perkaitomi, vadinami dešifravimu. Paprasčiausias šifravimo – dešifravimo pavyzdys iliustruotas 2.1 paveiksle.



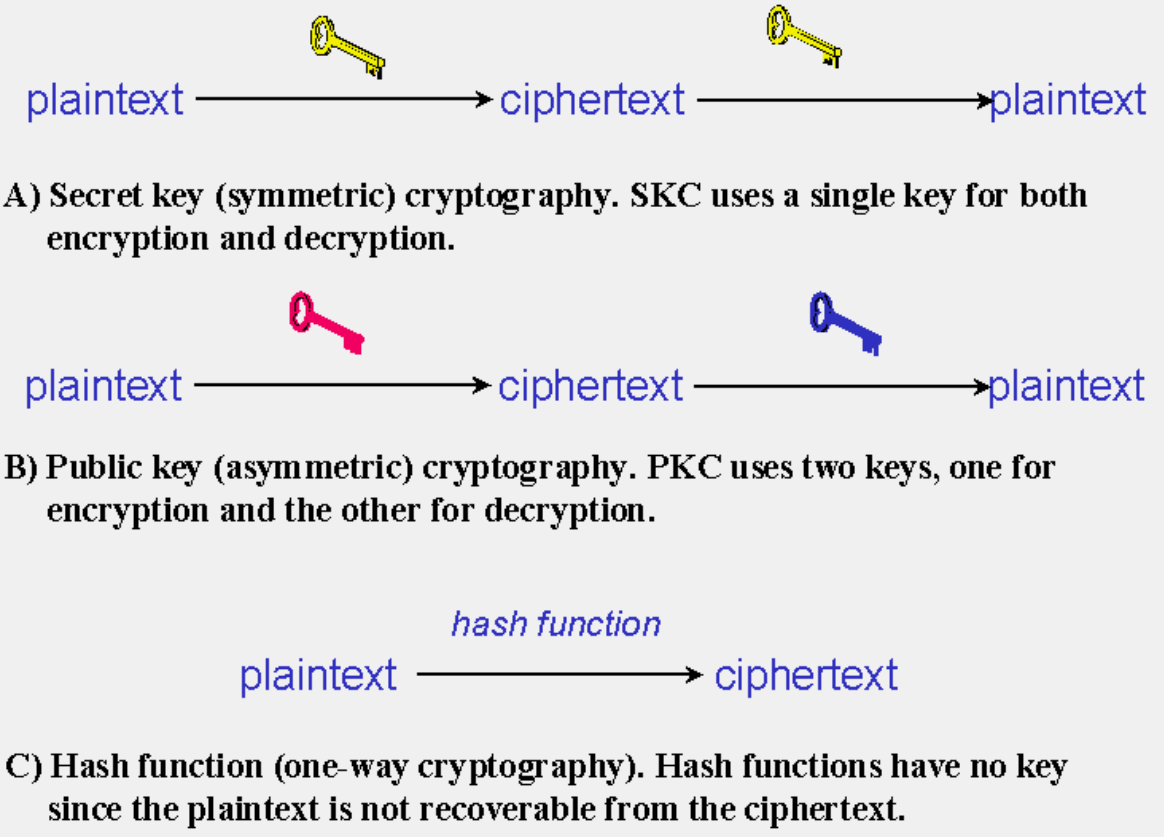
2.1 pav. Šifravimo – dešifravimo procesas.

Šifruoti duomenis žmonės pradėjo dar prieš mūsų erą. Vienas iš tokių buvo Julius Cezaris, Romos karvedys (100 m. pr. m. e. – 44 m. pr. m. e.). Jo šifravimo metodas buvo labai primityvus: savo laiško kiekvieną raidę pakeisdavo kita raide iš abėcėlės, pastumta keletu narių į kairę.

Informacijos šifravimas ypač svarbus tapo antrojo pasaulinio karo metais. Perduodant duomenis sąjungininkams ir kareiviams su tolimesniais nurodymais reikėjo saugotis kad jų neperimtų priešas ir neužkirstų kelio numatytai strategijai. Viena iš populiaresnių kodavimo technologijų išvystė vokiečiai. Tai buvo elektro-mechaninė mašina „ENIGMA“ [5]. Ji atrodė panaši į spausdinimo mašinėlę, tačiau rašant ją raidės būdavo perstumiamos kelėtos rotorių per tam tikrą skaičių. Tokį tekstą perskaityti galėjai tik turėdamas tokią pačią mašiną.

Atsiradus kompiuteriams žmonės ėmė keisis informacija skaitmeninėje erdvėje. Taigi iškilo būtinybė apsaugoti savo siunčiamą informaciją. Kompiuterinėje kriptografijoje yra sukurta keletas metodų duomenims saugoti. Dažniausia tokie metodai skirstomi į rūšis, pagal tai, kiek slaptažodžių reikia informacijai užšifruoti. Yra trys rūšys:

1) Slapto rakto algoritmai (naudojamas vienas raktas).
2) Viešo rakto algoritmai (naudojami du raktai).
3) Maišymo funkcijos (nenaudojamas joks slaptažodis).



2.2 pav. Šifravimo algoritmų rūšys. [6]

2.1 SLAPTO-SIMETRINIO RAKTO ALGORITMAI

Slapto rakto algoritmuose informacijai užkoduoti ir dekoduoti naudojamas vienas ir tas pats slaptažodis. Šiuo atveju skaitmeninė informacija yra sumaišoma pagal tam tikras matematines funkcijas kartu su raktu ir gaunamas šifruotas tekstas. Tada duomenys siunčiami gavėjui ir šis panaudojąs tokį patį slaptažodį dekoduoja informaciją. Kadangi čia yra naudojamas vienas raktas, šitos rūšies algoritmai dar vadinami simetrinio rakto algoritmais. Didžiausia simetrinio rakto algoritmų problema yra slaptažodžio perdavimas.

Slapto rakto šifrai dar skirstomi į srautinio tipo ir blokinius šifrus. Srautinio tipo algoritmuose duomenys šifruojami imant po vieną baitą ir sudedama xor funkcija su raktu. Po to, pagal tam tikras, iš anksto apibrėžtas matematines funkcijas, sugeneruojamas naujas raktas ir

sudedamas xor su sekančiu informacijos baitu ir t.t. Blokiniuose šifruose informacijos duomenys imami blokais(masyvais) po keletą baitų (pvz.: 56, 128 ir panašiai) ir tuomet jie perskaičiuojami pasitelkus matematines funkcijas bei logine operacija xor su raktu.

2.1.1 SRAUTINIAI ŠIFRAI

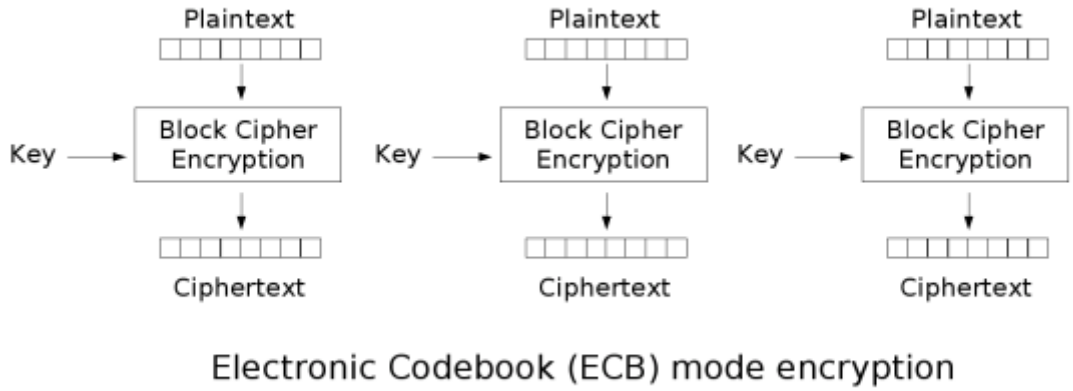
Srautinio tipo šifrai savo ruožtu dar skirstomi į sinchronizuojamuosius ir sinchronizuojančiuosius (*self-synchronizing*). Pagrindinis (ir vienintelis) skirtumas tarp jų yra tas, kad sinchronizuojamųjų šifrų raktų generacija priklauso nuo prieš tai buvusio šifruojamojo (arba dešifruojamojo) baito, o sinchronizuojančiųjų raktų generacija nepriklauso. Šiuo atveju, jeigu yra naudojamas sinchronizuojamasis šifras ir dešifruojant failo duomenis, jame yra išimta ar pridėta arba pakeista keletas baitų, sugeneruojami raktai taps kitokie negu kad buvo užšifruojant šį baitą ir daugiau jo nebe bus galima dešifruoti. Tuo tarpu sinchronizuojantieji šifrai dekoduos tuos failo baitus, kurie nebus pakeisti (tarsi dekoduotų teisingai dalį failo).

2.1.2 BLOKINIAI ŠIFRAI

Blokiniai šifrai taip pat turi keletą skirtingų modifikacijų, kaip užšifruoti failą.

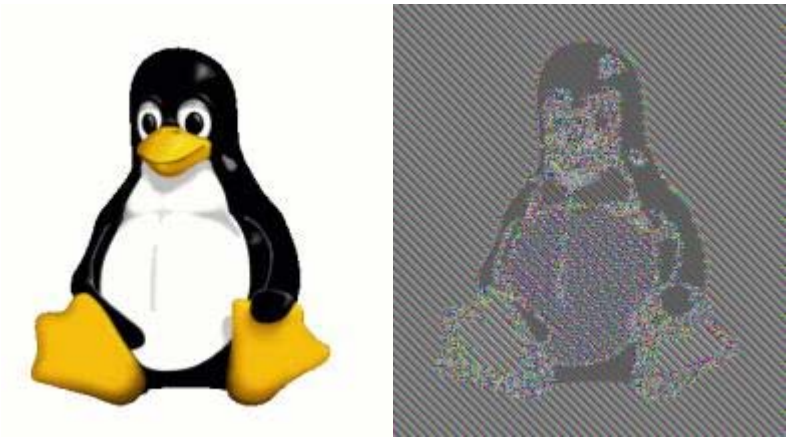
Pirmiausia reiktų pastebėti kad naudojant blokinius šifrus dažnai galima sutikti terminą pradinis vektorius (initialization vector) arba IV. Šis pirminis vektorius yra naudojamas pačioje šifravimo pradžioje ir turi būti tokio dydžio, kokio yra šifruojamieji blokai (pvz.: 128 bitai). Šiuo atveju prasidėjus šifravimo algoritmui, jis pirmiausia pasiims ne baitų bloką iš failo, bet IV ir nuo jo pradės šifravimo procesą. Jis naudojamas saugumui padidinti.

Elektroninė kodų knyga (*Electronic Codebook*) arba ECB operuoja tokiu principu. Kiekvienas duomenų blokas (tarkim 128 bitų) yra paimamas iš failo atskirai ir jam pritaikomas sugeneruotas raktas xor operacija (gali būti dar pridėta matematinių funkcijų). 2.3 paveiksle pavaizduotas ECB metodas. [7]



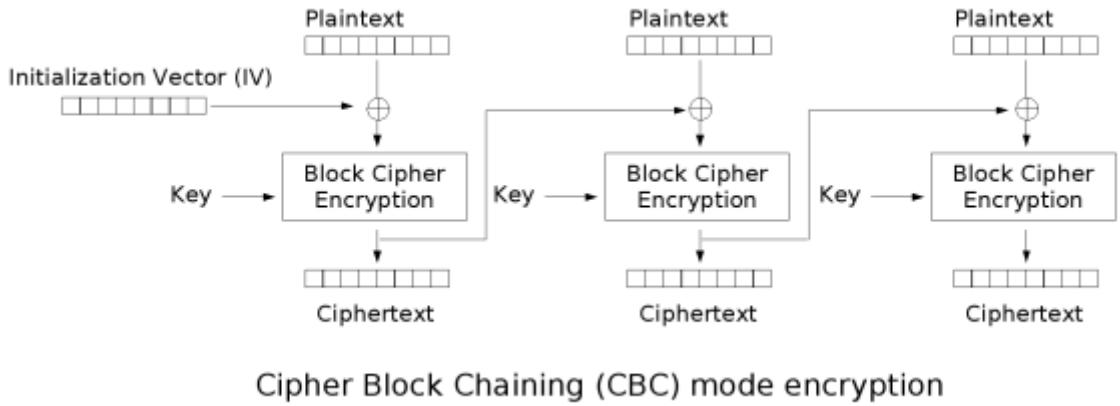
2.3 pav. ECB metodo schematinis paveikslas.

Šio metodo trūkumas yra toks, kad esant dviems tokiems patiems šifruojamiesiems bitų blokams išeityje atsiras du vienodi užšifruoti blokai. Pasinaudojęs šia savybę, įsilaužėlis/dešifruotojas gali bandyti atrasti tokius blokus ir iš jų surasti raktą. Šio metodo nepaartina naudoti. Esant tam tikroms sąlygoms, kartais netgi užšifravus failą, iš jo galima aiškiai matyti kaip atrodytų dešifruotas failas. Tai iliustruoja paveikslėlis 2.4



2.4 pav. Failas užšifruotas ECB metodu (šaltinis [8])

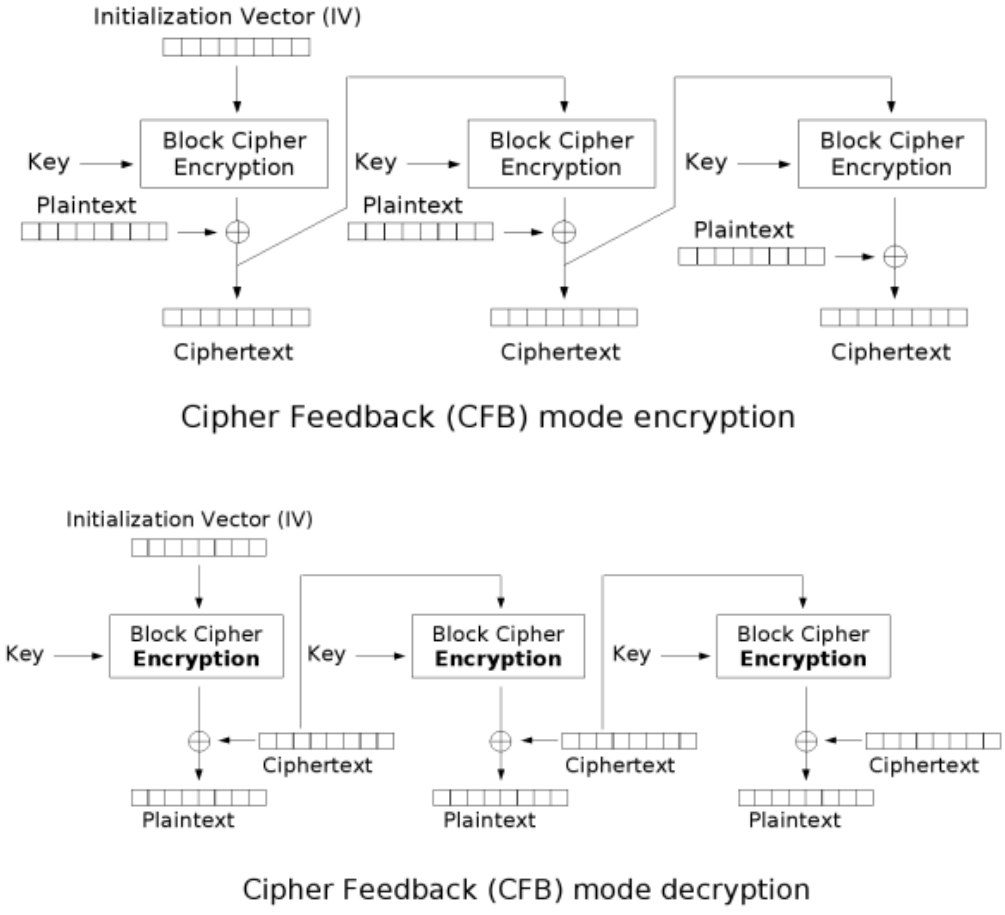
Grandininio šifravimo metodas *(Cipher Block Chaining)* arba CBC skiriasi nuo ECB tuo, kad užšifruojamas duomenų blokas sudedamas logine xor operacija su praeitu, jau užšifruotu duomenų bloku. Pats pirmasis duomenų blokas, saugumui padidinti taip pat sudedamas xor operacija su inicializacijos vektoriumi arba IV. Tokį metodą iliustruoja 2.5 paveikslas.



2.5 pav. CBC blokinio šifravimo metodo iliustracija. [7]

Dešifravimas vyksta atvirkštine tvarka: tarpinė būsena yra sudedama xor operacija su praeitu, jau dešifruotu duomenų bloku ir sugeneruotu raktu.

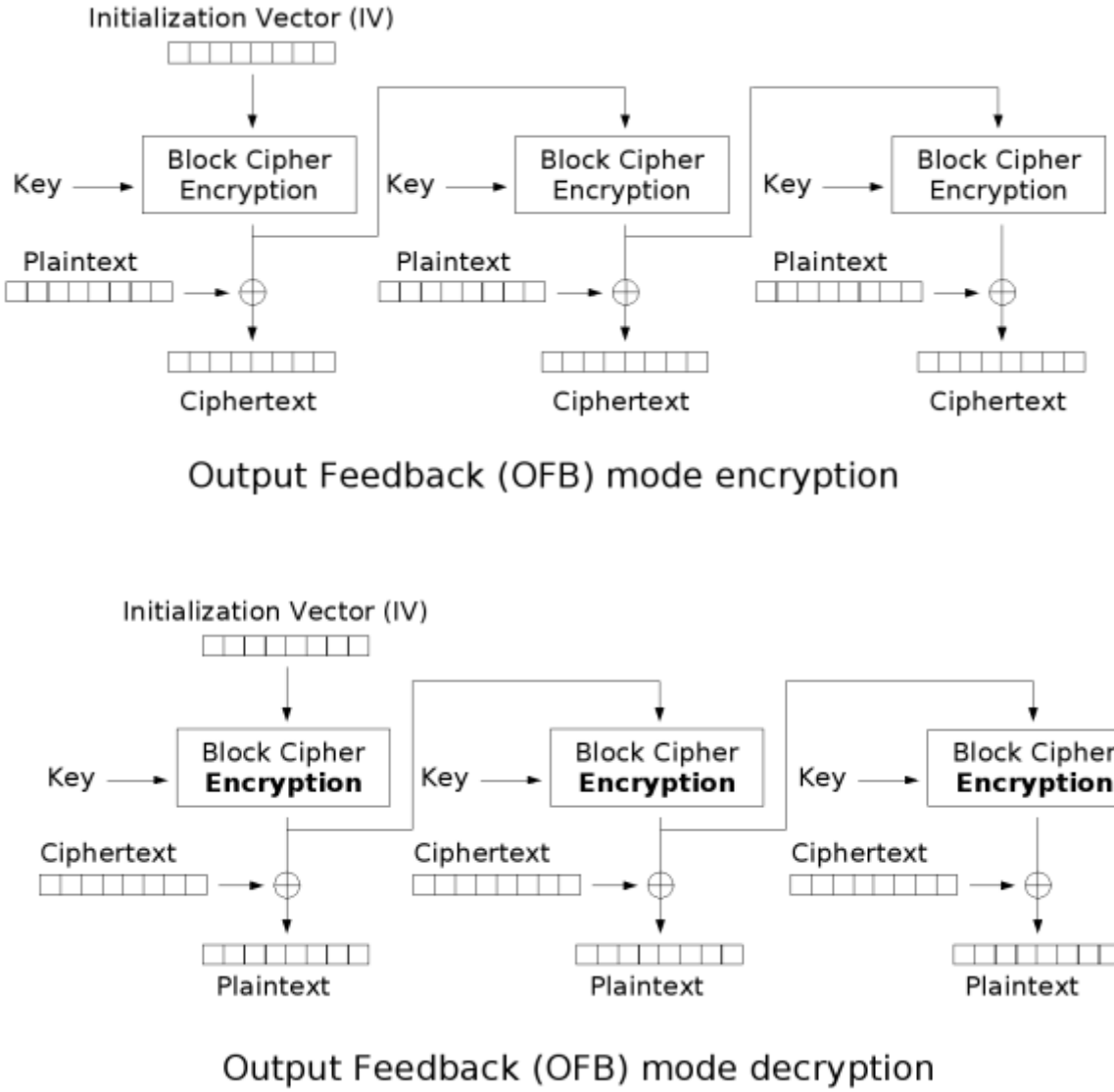
Grįžtamojo ryšio (*cipher feedback*) metodas arba CFB schema atvaizduota 2.6 paveiksle.



2.6 pav. CFB šifo metodo šifravimo ir dešifravimo schematinis atvaizdavimas. [7]

Grįžtamojo ryšio metodas iš esmės labai panašus į grandininio šifro metodą. CFB atveju pirmiausia yra vykdomas šifravimo procesas, panaudojus raktą bei praeitame etape užšifruotą duomenų bloką (arba pradinį vektorių, jei tai šifravimo pradžia) ir tik tuomet šis „kratinys“ jungiamas logine xor operacija su aiškiu (šifruojamuoju) tekstu.

Šifruoto grįžtamojo ryšio metodas (*output feedback*) arba OFD yra labai panašus CFB. Jo užšifravimo ir dešifravimo schema atvaizduota 2.7 paveiksle.



2.7 pav. OFD metodas [7]

OFD metodas nuo CFB skiriasi tik tuo, kad šifravimo etapuose vietoj jau užšifruoto bloko CFB metode naudojama tarpinės būsenos duomenys.

Vienas iš populiaresnių blokinių šifrų algoritmų buvo DES („buvo“ todėl, kad jį pakeitė AES). Šis algoritmas buvo sukurtas 1970 – aisiais metais kompanijos IBM. Nuo 1977 metų jis buvo

pripažintas tinkamu naudoti JAV vyriausybiniams duomenims šifruoti. Šis algoritmas dirba su 64 bitų blokais ir naudoja 56 bitų raktą jiems užšifruoti.

Naujausias ir patikimiausiu laikomas blokinis šifras AES (*advanced encryption standard*) . (žr. priedas)1997- aisiais metais JAV vyriausybė paskelbė konkursą naujam šifravimo algoritmui kurti, kad pakeisti atgyvenusį DES ir 2000 - aisiais metais du belgų šifruotojai: Joan Daemen ir Vincent Rijmen . Nuo jų pavardžių pirmųjų raidžių (*Rijndael*) buvo pavadintas algoritmas raktams generuoti.

2.2 VIEŠOJO RAKTO METODAS

Viešojo rakto šifravimo principas remiasi duomenų užšifravimu ir jų dešifravimu naudojant du skirtingus slaptažodžius. Šis metodas pirmą kartą paminėtas 1976 – aisiais metais Stanford'o universitete [6]. Tokiu būdu nebereikia perdavinėti slaptažodžio kitiems asmenims, kuriems siunčiama informacija nesaugia terpia. Šis metodas remiasi taip vadinama vienakryptėmis funkcijomis (*one-way functions*) .

Vienakryptės funkcijos, tai tokios matematinės funkcijos, kurios naudoja keletą kintamųjų rezultatui gauti. Tačiau turint rezultatą labai sunku sužinoti kintamuosius. Pavyzdžiui turint du skaičius 9 ir 16 juos sudauginus gaunasi 144, bet turint 144 nėra taip paprasta pasakyti kokie bus du sveikieji skaičiai, kuriuos sudauginus gausis 144.

Taigi viešojo rakto metoduose naudojami du skirtingi slaptažodžiai: vienas duomenims užšifruoti, kitas jiems dešifruoti. Šis metodas yra labai patogus, nes vieną raktą galima padaryti viešai prieinamą visiems ir bet kas, norintis perduoti jums slaptus duomenis, gali užšifruoti juos viešai prieinamu raktu, o juos dešifruoti galėsite tik jūs, nes tik jūs turėsite antrąjį raktą, reikalingą dešifravimui įvykdyti. Tokiu principu veikiantys algoritmai dar vadinami asimetrinė kriptografija.

Dargi vienas niuansas naudojant viešojo rakto algoritmus yra tas, kad juo galima įrodyti kas siuntė užšifruotus duomenis. Tarkim žmogus užšifravo žinutę savo privačiuoju raktu ir juos išsiuntė. Jeigu tą žinutę galima dešifruoti panaudojus jo viešąjį raktą, reiškia būtent tas asmuo ir siuntė tą žinutę.

Šiandien naudojamas vienas populiaresnių viešojo rakto algoritmų vadinamas RSA (nuo autorių pirmųjų pavardžių raidžių: Ronald Rivest, Adi Shamir, and Leonard Adleman). RSA metoduose naudojami raktai yra sudaromi iš labai didelio skaičiaus jį suskaidant į du mažesnius pirminius skaičius, tačiau irgi labai didelius (>100 skaitmenų), kurių sandauga būtų tas skaičius. Privatus ir viešasis raktai sudaromi per keletą žingsnių:

1) Paimkime du pirminius skaičius p ir q ir sudauginę juos gauname jų sandaugą n.

2) Išsirinkime trečią skaičių e, kuris būtų taip pat pirminis ir nesidalinti iš rezultato gauto iš (p-1)(q-1). Šis skaičius e bus viešojo rakto eksponentė.

3) Išskaičiuokime sveikąjį skaičių d iš reiškinio: (ed-1)/(p-1)(q-1) = sveikasis skaičius. Šis skaičius d bus privataus rakto eksponentė.

Taigi viešasis raktas bus (n, e) skaičių pora. Norint užšifruoti žinutę M su viešuoju raktu ir sukurti koduotą žinutę C naudojama lygtis: $C = M^e \bmod n$. Dešifravimas vyksta atvirkščiai: $M = C^d \bmod n$.

Iš pirmo žvilgsnio gali atrodyti, kad išskaičiuoti privatų raktą iš viešojo žinant skaičiavimo algoritmą yra paprasta, tačiau kuomet p ir q skaičiai yra labai dideli (>100 skaitmenų) skaičiavimai pasidaro sunkiai įmanomi. Sekančiame pavyzdyje parodomi e ir d skaičiavimai, tačiau kad būtų paprasta ir aišku viską suvokti, paimami p ir q skaičiai yra daug mažesni negu turėtų būti. Taigi:

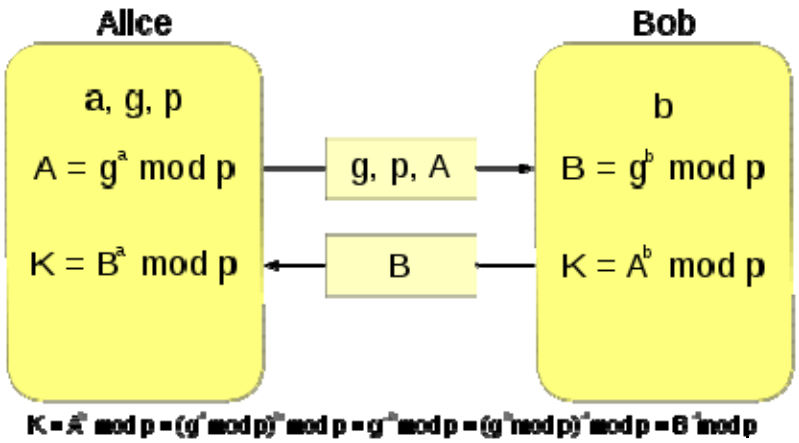
1) paimami p = 3, q = 5.
2) Gaunamas n = p * q = 15.
3) Skaičius e turi būti pirminis ir artimas sandaugai (p-1)(q-1). (2)*(4) = 8 . Taigi e = 11.
4) Skaičius d išskaičiuojamas iš reiškinio: (ed-1)/[(p-1)(q-1)] kurio rezultatas turi gautis natūralusis skaičius. Taigi (11d – 1)/8 . Vienas iš variantų d = 3.
5) Tarkim reikia užšifruoti ir išsiųsti žinutę „**SECRET**“ . Šito žodžio kiekvieną simbolį perverčiam į ASCII kodą. „SECRET“ = **83 69 67 82 69 84**
6) Siuntėjas užšifruoja kiekvieną skaitmenį atskirai pagal $C_i = M_i^{11} \bmod 15$. Į šitą formulę paduodami duomenys **836967826984** , o išeitis bus lygi **2c696d286924 .**
7) Gavėjas dešifruoja gautus **2c696d286924** su savo privačiu raktu (d,n) = (3,15) pagal formulę $M_i = C_i^3 \bmod 15$ ir gaunama išeitis yra **836967826984.** Šiuos skaičius perverčiam į simbolius pagal ASCII standartą ir gaunamas pradinis žodis „SECRET“

Šiame pavyzdyje buvo naudojami maži skaičiai (p = 3, q = 5), kad parodyti veikimo principą. Realiai naudojami raktų ilgiai yra 1024 ar 2048 bitų.

Dar vienas viešojo rakto naudojamų algoritmų yra Diffie-Hellman [9] sukurtas 1976 metais kryptoanalitikų Diifie ir Hellman, dar sutrumpintai vadinamas DH76. Šio algoritmo esmė yra perduoti slaptą raktą nesaugiu kanalu kitam asmeniui, kad pastarasis galėtų iššifruoti jam siustą žinutę. Protokolas turi du parametrinius skaičius p ir q. Šie skaičiai yra abudu viešai žinomi. Skaičius p yra pirminis, o g yra toks sveikasis skaičius, kad tenkintų tokią sąlygą: kiekvienam skaičiui n, tarp 1 ir p-1 imtinai egzistuoja toks laipsnis k skaičiaus g, kad sekanti lygybė yra patenkinama: $n = g^k \bmod p$.

Įsivaizduokite, kad Petriukas ir Onutė nori perdavinėti vienas kitam slaptažodį nesaugiu kanalu, pasinaudodami DH76 algoritmu. Taigi Onutė sugalvoja tik jai vienai žinomą skaičių a , o

Petriukas tik jam vienam žinomą skaičių b. Dabar jie vienas kitam perduoda savo skaičius. Tuomet kiekvienas iš jų pasinaudodami šiuo algoritmu, susigeneruoja reikšmę n. Onutės atveju tai bus $A = g^a \bmod p$, o Petriuko – $B = g^b \bmod p$. Tuomet jie perduoda vienas kitam šias naujai sugeneruotas n reikšmes ir išskaičiuoja bendrą slaptą raktą k, kuris lygus $k = A^B \bmod p$ ir $k = B^A \bmod p$. Šis algoritmas iliustruotas 2.8 paveiksle.



2.8 pav. DH76 raktų apsikeitimo principas. [10]

2.3 DUOMENŲ MAIŠYMO FUNKCIJOS

Duomenų maišos funkcijos – tai tokie užšifravimo algoritmai, kurių dešifruoti yra neįmanoma. Šie metodai paima duomenų bloką ir užšifruoja jį naudodami atsitiktiniai sugeneruotus skaičius, kaip parametrus maišymui. Taip gaunamas fiksuoto ilgio seka. Jos negalima modifikuoti(nes negalima iššifruoti) ar rasti kitą tokią pačią seką. Tokie algoritmai naudojami operacinių sistemų slaptažodžiams užkoduoti arba skaitmeniniams parašams įdiegti, tokiems patiems (besidubliuojantiems) įrašams duomenų lentelėse atrasti. Populiaresni duomenų maišos algoritmai yra: MD5 (*Message-Digest algorithm 5*) – generuojantis 128 bitų seką, SHA-1 (*Secure Hash Algorithm*) – generuojantis 160 bitų seką, atvirojo kodo RIPEMD-160 (RACE Integrity Primitives Evaluation Message Digest) taip pat generuojanti 160 bitų seką.

3. SHA-1

Vienas iš pirmųjų duomenų maišymo algoritmų buvo SHA-1 (*Secure Hash Algorithm*) sukurtas 1995 metais NSA (*national security agency*) agentūros [11]. SHA-1 algoritmas užšifruoja M žinutę, kuri yra *l* bitų ilgio. Čia *l* gali įgyti reikšmes $0 <= l < 2^{64}$. SHA-1 algoritmas sugeneruoja 160 bitų ilgio užšifruotą žinutę, naudodamas trijų rūšių kintamuosius:

1) aštuoniasdešimt žodžių, W_0, W_1, ... W_{79}, kurių kiekvienas sudaro 32 bitai funkciją;
2) penkių kintamųjų kombinaciją, a, b, c, d, e, kurių kiekvienas užima taip pat 32 bitus;
3) maišymo rezultatą, kuris laikomas penkiuose žodžiuose, $H_0^{(i)}$, $H_1^{(i)}$, $H_2^{(i)}$, $H_3^{(i)}$, $H_4^{(i)}$ po 32 bitus.

SHA-1 algoritmas naudoja eilę funkcijų $f_0, f_1, f_2, ..., f_{79}$. Kiekviena šių funkcijų operuoja su trejais 32 bitų žodžiais: x, y, z ir sugeneruoja 32 bitų ilgio žodį kaip rezultatą. Funkcija f_t (0<= t < 79) apibrėžiama sekančiai:

$f_{0\text{-}19} = Ch(x,y,z) = (x \square y) \oplus (\neg x \square z)$

$f_{20\text{-}39} = Parity(x,y,z) = x \oplus y \oplus z$

$f_{40\text{-}59} = Maj(x,y,z) = (x \square y) \oplus (x \square z) \oplus (y \square z)$

$f_{60\text{-}79} = Parity(x,y,z) = x \oplus y \oplus z$

SHA-1 algoritmas dar naudoja laikiną žodį T, kuris laiko tarpines maišymo funkcijų reikšmes ir perduoda jas penkiems $H_0^{(i)}$, $H_1^{(i)}$, $H_2^{(i)}$, $H_3^{(i)}$, $H_4^{(i)}$ žodžiams, kol galiausiai gaunasi bendras rezultatas H^N . Prieš prasidedant SHA-1 algoritmui, šifruojama žinutė turi būti tinkamai paruošta, t.y. ji yra papildoma iki reikiamo dydžio, remiantis žinutės papildymo procesu (žr. Priedas Žinutės užpildymas). Po to visi gauti duomenys suskirstomi į N blokų po 512 bitų: M^1, M^2, M^3, ..., M^N. Tuomet kiekvienam tarpinį maišymo rezultatą laikantiems žodžiams yra nustatoma pradinė reikšmė. Ši reikšmė yra nustatyta iš anksto ir kiekvienam algoritmui yra vis kitokia. Būtent SHA-1 algoritmui yra priskiriamos tokios reikšmės:

$H_0^{(0)}$ = 67452301

$H_1^{(0)}$ = efcdab89

$H_2^{(0)}$ = 98badcfe

$H_3^{(0)}$ = 10325476

$H_4^{(0)}$ = c3d2e1f0.

Prasideda maišymo procesas:

For i = 0 to N

{

1) Priskiriamos reikšmės žodžiams:

$$W_t = \begin{cases} M^{i}{}_{t} & 0 <= t < = 15 \\ ROTL(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) & 16 <= t < = 79 \end{cases}$$

2) Priskiriamos reikšmės penkiems (a,b,c,d,e) kintamiesiems:

$a = H_0^{i-1}$

$b = H_1^{i-1}$

$c = H_2^{i-1}$

$d = H_3^{i-1}$

$e = H_4^{i-1}$

4) For t= 0 to 79

{

$T = Rotl^5(a) + f_t(b,c,d) + e + K_t + W_t$

e = d

d = c

$c = Rotl^{30}(b)$

b = a

a = T

}

5) Suskaičiuojama i -oji tarpinė maišymo reikšmė $H^{(i)}$:

$H_0^{(i)} = a + H_0^{(i-1)}$

$H_1^{(i)} = b + H_1^{(i-1)}$

$H_2^{(i)} = c + H_2^{(i-1)}$

$H_3^{(i)} = d + H_3^{(i-1)}$

$H_4^{(i)} = e + H_4^{(i-1)}$

}

Kai visas ciklas pakartojamas N kartų, tuomet paskutinės H reikšmės pridedamos (ne sudedamos kaip matematinė sudėtis, bet sulipdomos) ir gaunasi H^N rezultatas.

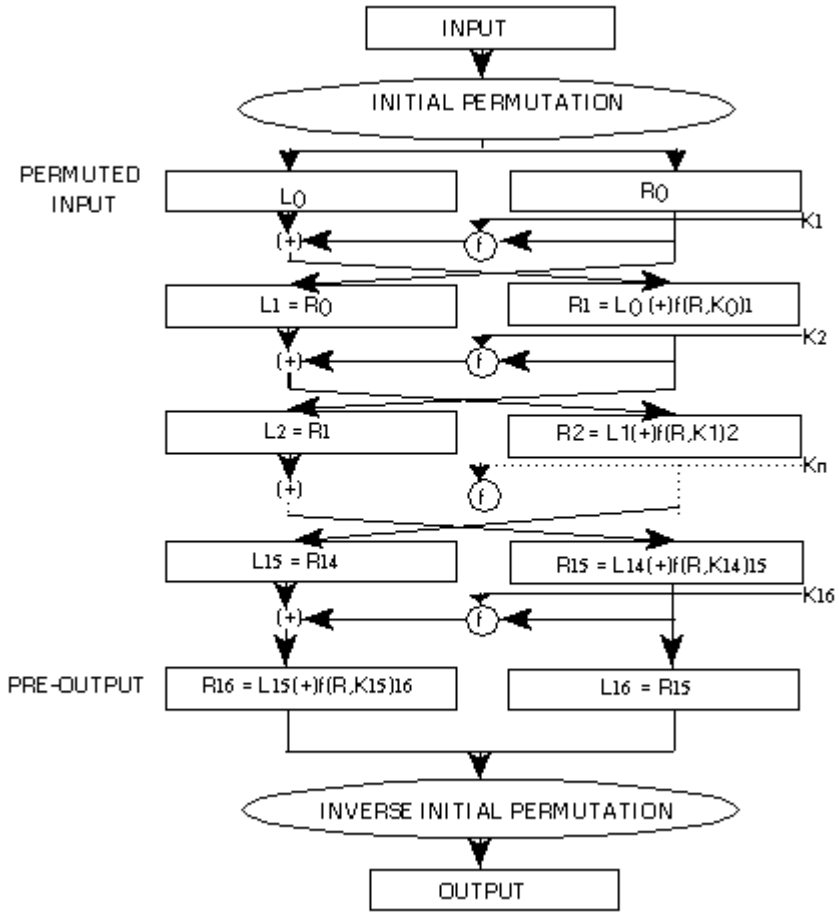
$H^N = H_0^N \square H_1^N \square H_2^N \square H_3^N \square H_4^N$.

SHA-1 žinutės šifravimo pavyzdys parodytas priede.

4. DES ŠIFRAVIMO STANDARTAS

DES (*data encryption standard*)– tai šifravimo algoritmas sukurtas NIST [12] (*National Institute of Standards and Technology*) užsakymu 1976, saugoti JAV valstybinės svarbos skaitmeninę informaciją. Ilgą laiką sklandė gandai kad NSA (*national security agency*) galėjusi nulaužti DES standartą pasitelkusi savo galingą aparatinę įranga.

Algoritmas sukurtas užšifruoti duomenis naudojant 64 bitų raktą ir operuojant su 64 bitų duomenų bloku. Dešifravimas vykdomas su tuo pačiu raktu, bet operacijos išdėstomos atvirkščia tvarka nei šifruojant. Blokas, perduotas šifravimui, pereina tris etapus: bitų sukeitimo operaciją (žymėsime **IP**), nuo rakto priklausomą ciklą, kuris turi šešiolika etapų, ir atvirkščią bitų sukeitimo operaciją (žymėsime $\mathbf{IP^{-1}}$) . Pirmiausia duomenų blokas yra perduodamas IP operacijai, po to nuo rakto priklausomą skaičiavimų ciklą, kuris susideda iš funkcijos ***f,*** kuri vadinama šifro funkcija ir funkcijos ***KS,*** kuri yra raktų generavimo funkcija. DES operuoja su duomenimis, kurie yra paimami blokais po 64 bitus, tačiau antrame šifravimo etape jis šį bloką padalina į du blokus: kairį bloką (žymėsime L (nuo angliško žodžio left)) ir dešinį (žymėsime R), po 32 bitus kiekvienas. Taigi visas blokas bus LR, kur pirmasis bitas yra kairio bloko pirmasis bitas, po to seka kairio bloko antrasis bitas, po to trečiasis ir užsibaigia 32 -uoju dešinio bloko bitu. Visas šifravimo procesas atvaizduotas 4.1 paveiksle.



4.1 pav. DES šifravimo etapai [13].

IP transformacija paima 64 bitų bloką ir sukeičia jame esančius narius vietomis pagal sekančią lentelę.

4.1 lentelė. IP funkcijos transformacija.

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

Reiškia į pirmąją vietą bus įdėtas 58 –asis bitas, į antrąją į 50 –asis bitą, į trečiąją 42 –asis, į paskutinę – 7 –asis bitas. Po to šiaip modifikuotas 64 bitų duomenų blokas yra perduodamas

ciklinei transformacijai, kurioje dalyvauja funkcijos *f* ir *KS* . Šis blokas yra dalijamas į dvi dalis po 32 bitus. Kairysis blokas *L* turės narius nuo vieno iki trisdešimt antrojo bito, o dešinysis R likusius, t.y. nuo trisdešimt trečiojo iki šešiasdešimt ketvirtojo imtinai. Tuomet kiekviename ciklo etape iš bloko LR yra gaunamas blokas L‘R‘ tokiu būdu:

$L' = R$

$R' = L \oplus f(R,K)$

Čia K yra 48 bitų blokas, gaunamas pasinaudojus *KS* funkcija, su kintamaisiais n – natūralusis skaičius nuo 1 iki 16 ir šifro raktu KEY, kuris yra 64 bitų blokas.

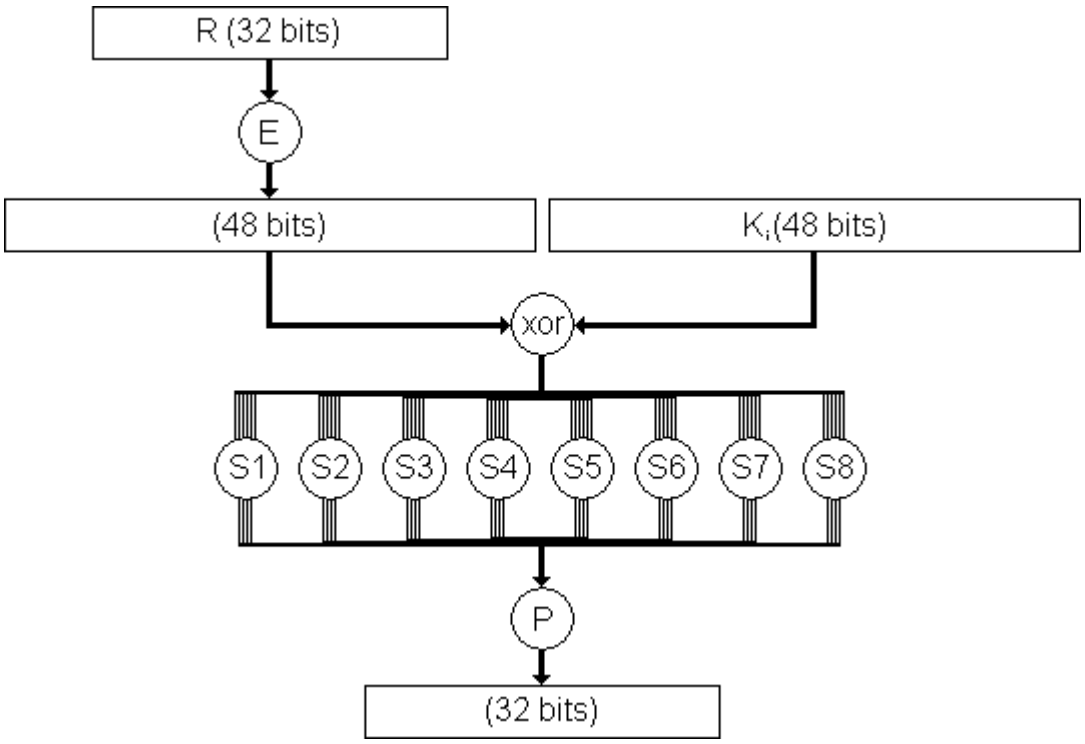
$K_n = KS\ (n, KEY)$

Natūralusis skaičius n atitinka iteracijos skaičių ir jis apsprendžia, nuo kurios pozicijos reikės paimti 48 bitus iš rakto 64 bitų rakto KEY. Taigi L ir R blokų formules galima papildyti:

$L_n = R_{n-1}$

$R_n = L_{n-1} \oplus f(R_{n-1}, K_n)$.

Funkcijos *f* veikimas atvaizduotas 4.2 –ame paveikslėlyje.



4.2 pav. F funkcijos veikimo schematinis atvaizdavimas.

Pirmiausia suveikia funkcija E. Ji paima dešinįjį 32 –jų bitų duomenų bloką ir padaro iš jo 48 duomenų bloką. Visas procesas vyksta panašiai kaip IP transformacija. Ji atvaizduota sekančioje lentelėje.

4.2 lentelė. ***E funkcijos transformacija.***					
32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	33

E funkcija ima dešiniojo duomenų bloko trisdešimt antrąjį narį ir formuodama naują, 48 bitų bloką, deda jį į pirmąją vietą, po to ima pirmąjį narį ir deda jį į antrąją vietą ir t.t. pagal 4.2 lentelę. Taip suformuojamas 48 bitų naujas blokas. Po to šis blokas sudedamas xor logine operacija su kitu, 48 bitų bloku K, gautu iš šifravimo rakto, pagal jau anksčiau minėtą funkciją KS.

Sekančiame etape prasideda S funkcijų veikimas. Jų iš viso yra aštuonios. Kiekviena iš S funkcijų gauna šešis bitus informacijos ir rezultate sugeneruoja naujus keturis bitus pagal 4.3 lentelę.

4.3 lenelė. S funkcijos generacinė lentelė.

No.	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

Funkcijų S veikimas pagal trečią lentelę apibrėžiamas taip. Į funkciją S paduodamas 6 bitų duomenų blokas. Pirmas ir paskutinis, šio bloko skaičiai reiškia eilutės skaičių lentelėje, o keturi vidurinieji bitai, reiškia stulpelio skaičių lentelėje. Būtent tas nurodytas skaičius ir bus funkcijos S rezultatas.

Pavyzdžiui: į funkciją $S_1(B)$ buvo paduotas blokas B = 011011. Taigi pirmasis bitas bus 0, o paskutinis – 1. Juos pridedant vieną prie kito gaunasi 01. Šis skaičius atspindi lentelės antrąją eilutę. Vidurinieji keturi bloko B bitai sudaro skaičių 1101, tai reiškia kad jie nurodo 13 stulpelį lentelėje. Taigi 2 eilutė ir 13 stulpelis rodo į skaičių 5, kuris ir bus funkcijos S sugeneruotas rezultatas.

Taip sugeneravus visus aštuonis S funkcijų rezultatus po 4 bitus, gaunasi 32 –jų bitų blokas. Šis blokas perduodamas sukeitimo operacijai P. Ji veikia lygiai taip pat kaip ir sukeitimo operacija IP, tik pagal 4.4 lentelę.

4.4 lentelė. P sukeitimo transformacija.			
16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

Pasibaigus šešiolikai ciklo etapų, kuriuose LR blokai buvo transformuojami pagal aprašytas funkcijas, rezultatas perduodamas IP^{-1} transformacija, kuri yra priešingybė IP transformacijai, apibrėžtai algoritmo pradžioje. IP^{-1} transformacija duomenų bloko bitus perdėlioję pagal specifikaciją, kuri yra atvaizduota 4.5 lentelėje.

4.5 lentelė. IP^{-1} transformacija.							
40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

Po šios bitų transformacijos, DES šifravimas su vienu, 64 bitų duomenų bloku yra baigiamas ir imamas sekantis blokas.

Dešifravimo procesas vyksta atvirkščia tvarka: pirmiausia užšifruotas duomenų blokas patenka į IP^{-1} transformaciją, po to eina 16 etapų ciklas, kuriuose

$R = L'$

$L = R' \oplus f(L',K)$ arba

$R_{n-1} = L_n$

$L_{n-1} = R_n \oplus f(L_n, K_n)$

ir galiausiai perduodamas IP transformacijai.

5. AES ŠIFRAVIMO STANDARTAS

5.1. ĮŽANGA

AES – pažangus šifravimo standartas (advanced encryption standard), skirtas elektroniniams duomenims užkoduoti. Šis algoritmas paima suprantamus duomenis ir perverčia į užkoduotus. 2001 -aisiais patvirtintas JAV standartu ir technologijų instituto kaip patikimas slaptiems vyriausybiniams duomenims saugoti. Šiuo metu jis laikomas patikimiausiu šifravimo algoritmu ir yra naudojamas daugelyje šalių ir nevyriausybinių organizacijų ypač slaptiems duomenims užkoduoti.

AES yra simetrinis blokinis šifruotojas, jo pirmtakas buvo DES (data encryption standard) . DES operavo su 64 bitų duomenų blokais ir buvo užšifruojamas su 56 bitų raktu. Dabartinis AES yra trijų atmainų. Jis operuoja su 128 bitų duomenų bloku ir jiems užšifruoti naudoja 128, 192 arba 256 bitų ilgio raktus. Pats svarbiausias AES saugumo parametras yra būtent rakto ilgis, su kurio pagalba ir bus užšifruojama informacija. Raktai gaunami pasitelkiant *Rijndael* algoritmus. Šį algoritmą sukūrė belgų kriptografijos specialistai Jonas Daemenas ir Vincentas Rijmenas (Joan Daemen ir Vincent Rijmen). Nuo jų pavardžių pirmųjų raidžių ir buvo pavadintas šis raktų generavimo algoritmas. Šis algoritmas iš vieno, atitinkamo ilgio slaptažodžio sugeneruoja daugiau slaptažodžių, kurie visame šifravimo fazėje yra naudojami skirtingiems duomenų blokams užšifruoti. Mano programa parašyta naudojant 128 bitų ilgio šifravimui raktą.

5.2. INFORMACIJOS ATVAIZDAVIMAS AES ALGORITME

AES šifras duomenis koduoja blokais. Šiame standarte toks blokas užima lygiai 128 bitus informacijos. Jis būna patalpintas masyvo tipo kintamajame, kurio pradinis indeksas yra 0 (nulis), o galinis vienetu mažesnis nei sekos narių suma. AES tokius blokus sudeda su etapo raktais, kurie priklausomai nuo AES rūšies gali būti skirtingų dydžių: 128, 192 arba 256 bitų informacijos. Šie raktai lygiai taip pat kaip ir blokai yra laikomi masyvuose. Mano rašytoje programoje yra naudojamas 128 bitų AES standartas. Masyvuose esantys duomenis visgi laikomi ne bitų pavidale, bet baitų. Baitas sudarytas iš aštuonių bitų. Taigi 128 bitų koduojamos informacijos blokas bus sudarytas iš masyvo, turinčio 16 narių ir žymimas a[n]. Kiekvienam iš trijų rūšių AES standarto, raktams laikyti bus skirtingo dydžio masyvai a[n] :128 AES 0 <= n < 16, 192 AES 0<=n<24 it 256 AES 0<=n<32. Nors blokų ir raktų atvaizdavimas yra baitų pavidale, tačiau kompiuterio

procesorius visus skaičiavimus atlieka dvejetainėje sistemoje arba kitaip tariant į informaciją žiūri kaip į bitų seką, todėl reikėtų išmokti atvaizduoti baitą kaip bitų sekas.

Vienas iš būdų bitams atvaizduoti, juos vaizduojant kaip polinomą, sudarytą iš kintamųjų ir jų koeficientų. Tokiu atveju baitą sudarys seka: $b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0x^0 = \sum b_ix_i$. Pavyzdžiui baitas 01100011 taps daugianariu $b_6x^6 + b_5x^5 + b_1x^1 + b_0x^0$.

Kitas patogus būdas yra pasiremti šešioliktaine skaičiavimo sistema. Kiekviena baitą sudaro 8 skaitmenys iš dvejetainės sistemos arba 2 iš šešioliktainės. Tokiu vienas šešioliktainės sistemos skaitmuo sudarys pusbaitį. Šešioliktainė sistema labai patogi verčiant į ir iš dvejetainės.

5.1 lentelė. Šešioliktainės ir dvejetainės sistemos palyginimas

Bitų seka	Šešioliktainis simbolis	Bitų seka	Šešioliktainis simbolis	Bitų seka	Šešioliktainis simbolis	Bitų seka	Šešioliktainis simbolis
0000	0	0100	4	1000	8	1100	C
0001	1	0101	5	1001	9	1101	D
0010	2	0110	6	1010	A	1110	E
0011	3	0111	7	1011	B	1111	F

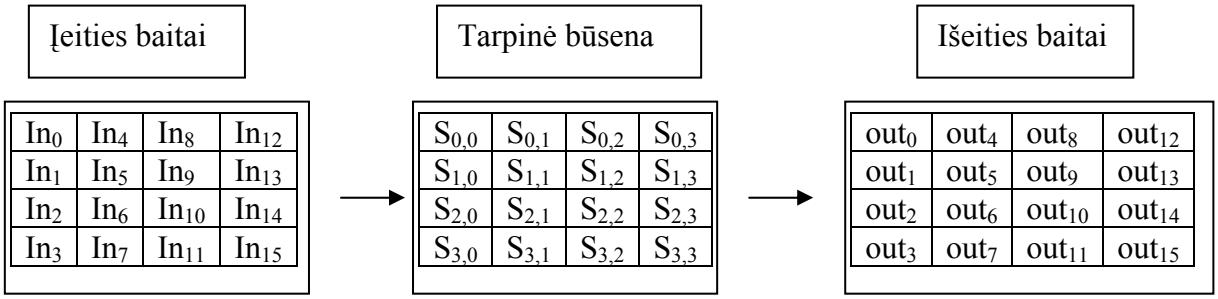
Taigi bitų seką 01100011 galima atvaizduoti kaip šešioliktainį skaičių {63}. Patikslinimui: 0110 būtų 6 , o 0011 – 3, tik reikia pabrėžti, kad kairinis pusbaitis reiškia dešimtis, o dešininis – vienetus.

AES duomenų šifravimo metu, tam tikrose situacijose, naudoja ir iš 9 skaitmenų sudaryta bitų seką, jai atvaizduoti iš polinomo pusės priekyje pridedamas b_8x^8 , o iš šešioliktainės vaizdavimo pozicijos, priekyje pridedamas {01}.

AES duomenis priima ir atiduoda dvimačio masyvo pavidale. Kai duomenys šiame pavidale yra šlifuojami, tokia būsena dar vadinama tarpine. Ją sudaro 4 eilutės ir NB stulpeliai. Šiame standarte NB yra lygus 4. Taigi tarpinė būsena bus sudaryta iš 16 nariu, kurių kiekvienas turi sau būdinga adresą ir yra vaizduojamas S[r,c] : r- eilutės skaičius, c – stulpelio.

Paprasčiau atvaizduoti AES veikimą 5.2 lentelėje.

5.2 lentelė. AES veikimo principas.



Input bytes – tai pradiniai duomenys.

State array – šifruojami pradiniai duomenys.

Output bytes – užkoduoti duomenys.

Galima matyti, kad prieš šifravimo procesą, duomenys paimami iš failo kaip vientisa eilė: in0, in1, in2 ..... in15 ir tik tada pervedami į dvimatį masyvą tokia tvarka: S[r,c] = in[r+4c] . Kai duomenų masyvas baigiamas šifruoti, jis vėl pervedamas į vienmatį masyvą ir imama nauja duomenų porcija. Taip iš norimo užšifruoti failo vis imama 16 baitų informacijos porcija, pervedama į 4x4 dvimatį masyvą, užšifruojama ir pervedama atgal į failą.

Tarpinę būseną, o taip pat ir pradinius bei šifruotus duomenis dvimačiuose masyvuose dar galima laikyti kaip keturis 32 bitų žodžius (žr. Terminai „*žodis*“). Tokie žodžiai sudaromi sudedant masyve esančius narius su tais pačiais stulpelių indeksais. Šiuo atveju, iš pirmajame paveiksle tarpinė būsena būtų:

$W_0 = S_{0,0}\, S_{1,0}\, S_{2,0}\, S_{3,0}$

$W_1 = S_{0,1}\, S_{1,1}\, S_{2,1}\, S_{3,1}$

$W_2 = S_{0,2}\, S_{1,2}\, S_{2,2}\, S_{3,2}$

$W_3 = S_{0,3}\, S_{1,3}\, S_{2,3}\, S_{3,3}$

5.3. MATEMATINĖS OPERACIJOS

Kompiuterio procesorius su duomenimis elgiasi ne kaip su paprastais dešimtainiai skaičiais, o kaip su bitų seka. Tokia bitų seka atvaizduojama kaip daugianaris (polinomas) ir vadinama *Galua* lauku (*Galois Field*): žymima $GF(q^n)$. *Galua* laukas – tai seka, susidedanti iš n+1 narių, kurių pagrindas yra q ir laipsnis n. Su tokiais daugianariais AES standartas atlieka įvairius matematinius veiksmus, tačiau jie truputi skiriasi nuo mums įprastų su dešimtainiais skaičiais.

5.3.1. SUDĖTIS

Norint sudėti du polinomus, reikia jų koeficientus, esančius prie tokio pačio laipsnio kintamojo lyginti logine operacija xor (ženklas $\oplus$) . Xor operacija: 1 xor 0 = 1 , 1 xor 1 = 0, 0 xor 0=0. Taigi sudedant $\{a_0\ a_1\ a_2\ a_4\ a_4\ a_5\ a_6\ a_7\}$ ir $\{b_0\ b_1\ b_2\ b_3\ b_4\ b_5\ b_6\ b_7\}$ suma bus $\{c_0\ c_1\ c_2\ c_3\ c_4\ c_5\ c_6\ c_7\}$ ir kiekvienam $c_i = a_i + b_i$.

Pavyzdžiui:

$(x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) = x^7 + x^6 + x^4 + x^2$

arba galima tokia sudėti atvaizduoti kaip bitų seką:

(01010111) + (10000011) = 11010100

arba kaip šešioliktainės sistemos skaičius:

{57} xor {83} = {D4} .

Polinomų sudėtis veikia analogiškai.

5.3.2. DAUGYBA

AES standarte dauginant du baitus (daugyba žymima ● ženklu), išreikštus polinomais, vykdoma kiekvieno jų nario daugyba su kito polinomo nariu, o po to vienos rūšies nariai yra sudedami ne įprastu „+“ (plius) veiksmu, bet xor. Vėliau, jeigu rezultatas gaunasi polinomas su nariais, kurių laipsniai didesni nei 7 , jam vėl vykdoma moduliacijos (žymėsime „**mod**“) operacija su pirminiu polinomu.

Moduliacijos operacija, tai tokia operacija, kuomet dviejų polinomų nariai surašomi stulpeliu nuo kairiausių (turinčių didžiausią laipsnį) į dešinę ir lyginami xor operacija esantys vienas virš kito.

Pirminiu polinomu vadinamas toks daugianaris, kuris nesidalina be liekanos iš nieko kito išskyrus vienetą ir save patį. Šiame AES standarte toks polinomas bus $x^8 + x^4 + x^3 + x + 1$. Išvertus į šešioliktainę skaičiavimo sistemą jis bus lygus {01}{1B} arba (100011011) dvejetainėje. Tarkim reikia sudauginti du baitus: {57}● {83} = {C1} .Tokia daugyba baigtinėje sekoje sutrumpintai galima užrašyti matematikos kalba: $(x^6 + x^4 + x^2 + x + 1) \bullet (x^7 + x + 1) \bmod (x^8 + x^4 + x^3 + x + 1)$. Visas daugybos procesas, kurį atlieka AES algoritmas, pavaizduotas detaliau:

$\{57\} = (x^6 + x^4 + x^2 + x + 1)$; $\{83\} = (x^7 + x + 1)$

$(x^6 + x^4 + x^2 + x + 1)\ (x^7 + x + 1) = x^{13} + x^{11} + x^9 + x^8 + x^7 + x^7 + x^5 + x^3 + x^2 + x + x^6 + x^4 + x^2 + x + 1$ = (dabar to pačio laipsnio narius sudedame operacija xor) $x^{13} + x^{11} + x^9 + x^8 + (1 \text{ xor } 1)x^7 + x^6 + x^5 + x^4 + x^3 + (1 \text{ xor } 1)x^2 + (1 \text{ xor } 1)x + 1 =$

$= x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1$ Kadangi gautas polinomas turi narių, kurių laipsniai didesni nei 7, todėl jis moduliuojamas su pirminiu polinomu. Taigi pastarąjį rezultatą perrašom į bitų seką: 10101101111001 ir mod su 100011011 :

10101101111001

100011011

00100000011001=$x^{11} + x^4 + x^3 + 1$, kartojam tol, kol nebeliks narių su laipsniais, aukštesniais nei 7.

100000011001

100011011

000011000001 = $x^7 + x^6 + 1$ Toks yra galutinis rezultatas. Pervedus į bitų seką : 11000001, pervedus į šešioliktainę sistemą: {C1}. Laipsnių mažinimas iki 7-ojo ir mažesnių laipsnių reikalingas tam, todėl, kad AES algoritmas operacijas vykdo su vieno baito dydžio nariais, o atliekant įvairius matematinius veiksmus, dažnai narių dydžiai išeina iš už baito ribų(tampa didesni nei vienas baitas), todėl juos reikia sumažinti. Visa šita daugybos operacija su polinomais AES standartui galima užrašyti sutrumpinta matematine išraiška:

$a(x) \bullet b(x) \bmod m(x)$.

5.3.3. DALYBA

Baigtinių skaičių sekoje dalyba yra vykdoma randant daugianariui atvirkštinį daugianarį ir jį dauginti. Tarkim pažymime daugianarius a(x) ir b(x). Atvirkštinis daugianaris daugianariui b(x) bus žymimas $b^{-1}(x)$. Taigi norint a(x) padalinti iš b(x) reikia sudauginti $a(x) \bullet b^{-1}(x) \bmod m(x)$.
Atvirkštinis daugianaris randamas pasinaudojus *Euklido*[14] algoritmu. Tereikia atlikti tokį ciklą:

```
liekana[1] := f(x)
liekana[2] := b(x)
kintamasis[1] := 0
kintamasis [2] := 1
i := 2
while liekana[i] > 1
   i := i + 1
   liekana[i] := remainder(liekana[i-2] / liekana [i-1])
   koeficientas[i] := quotient(liekana [i-2] / liekana [i-1])
   kintamasis [i] := - koeficientas[i] * kintamasis[i-1] + kintamasis[i-2]
atvirkštinis := kintamasis[i]
```

čia: f(x) – yra pirminis polinomas, su kuriuo vyksta moduliacijos operacija.

b(x) – daugianaris, kuriam reikia rasti atvirkštinį.

Remainder() – funkcija, kuri randa dviejų skaičių(šiuo atveju daugianarių) dalmens liekaną.

Quotient () – funkcija, kuri randa dviejų skaičių(šiuo atveju daugianarių) dalmenį, atmetus trupmeninius skaičius.

Šis ciklas yra vykdomas tol, kol liekana tampa lygi 1, jeigu ji niekada netampa 1, tuomet toks daugianaris neįmanomas. Pirmosios dvejos liekanos yra duotos - tai pirminis polinomas ir polinomas, kuriam atvirkštinį reikia rasti, taip pat pirmi du kintamieji duoti: 0 ir 1. Pavyzdžiui turim $f(x) = x^8 + x^4 + x^3 + x + 1$ ir $b(x) = x^6 + x^4 + x + 1 = \{53\}$, kuriam reikia rasti atvirkštinį, dedam juos i Euklido algoritmo ciklą ir sukam:

i	liekana	koeficientas	kintamasis
1	$x^8 + x^4 + x^3 + x + 1$		0
2	$x^6 + x^4 + x + 1$		1
3	x^2	$x^2 + 1$	$x^2 + 1$
4	$x + 1$	$x^4 + x^2$	$x^6 + \cancel{x^4} + \cancel{x^4} + x^2 + 1$
5	1	$x + 1$	$x^7 + x^6 + x^3 + \cancel{x^2} + \cancel{x^2} + x + \cancel{1} + \cancel{1}$

Taigi $b^{-1}(x) = x^7 + x^6 + x^3 + x = \{CA\}$.

5.4. ŠIFRAVIMO ALGORITMAS

AES algoritmas koduoja informaciją blokais. Šitie blokai yra paimami iš norimo koduoti failo, surašomi į dvimatį masyvą ir su jais atliekama keletas maišymo, perkėlimo ir sudėjimo operacijų. Kaip jau buvo minėta anksčiau, šitie blokai susideda iš 4 eilučių ir NB stulpelių. AES standartui NB = 4 , taigi blokai susideda iš 4x4 matricos, kurioje kiekvienas narys užima vieną baitą informacijos. AES skirstomas į tris porūšius: AES-128, AES-192 ir AES-256 . Kiekvienas iš pastarųjų skaičių nusako AES algoritme naudojamo slaptažodžio dydį bitais. Iš vieno pagrindinio slaptažodžio, tam tikru algoritmu yra sugeneruojama daug kitų raktų - etapo raktų. AES-128 sugeneruoja 4 pradinius raktus, AES-192 – 6, o AES-256 – 8 pradinius raktus (kitaip žodžius) , kurie yra naudojami šifravimo cikle ir kitų etapo raktų generavime. Taigi skaičius NK nusakys kiek pradinių žodžių yra sugeneruojama iš slaptažodžio. Dar vienas parametras, kuris skirsto AES į porūšius yra skaičius NR. Jis nusako kiek kartų duomenų blokas bus šifruojamas, t.y. kiek kartų

tarpinė būsena bus praleidžiama per AES algoritmo kodavimo ciklą. Jis vadinamas raundų skaičius. Apibendrinant AES porūšius:

	NK (raktų skaičius)	NB (stulpelių skaičius)	NR (raundų skaičius)
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

Kiekviename raunde, šifruojami duomenys (duomenų blokas) yra perduodamas keturioms kodavimo funkcijoms:

1. SubBytes – kiekvieną bloko narį pakeičia kitu, iš S-box‘o,.
2. ShiftRows – kiekvieną eilutę pastumia per tam tikrą skaičių į kairę.
3. MixColumns – kiekvieną stulpelį daugina iš tam tikro polinomo.
4. AddRoundKey funkcija, kuri prie esamų duomenų logine xor operacija prideda sugeneruotą etapo raktą.

Kuomet duomenis reikia dekoduoti, AES algoritmas kreipiasi į kitas, priešingas pastarosiom, funkcijas: InvSubBytes, InvShiftRows, InvMixColumns tik addRoundKey funkcija lieka tokia pati. Kodavimo algoritmas atvaizduotas sekančioje schemoje:

```
crypt(inByte as array)
dim state as array = inByte
Dim round As Integer = 0
        addRoundKey(state, round)
        For round = 1 To 9
           subBytes(state)
           shiftRows(state)
           mixColumns(state)
           addRoundKey(state, round)
        Next
        subBytes(state)
        shiftRows(state)
        addRoundKey(state, 10)
outByte = state
```

Kaip matyti 5.1 lentelėje, AES vykdo tokį algoritmą: pasiima iš failo duomenis kuriuos reikės šifruoti (*inByte*), priskiria tarpiniai būsenai (*state*), ir perduoda funkcijai addRoundKey. Šioje funkcijoje tarpinė būsena yra sudedama su pirmaisiais trejais sugeneruotais žodžiais arba pirmuoju etapo raktu. Po to, jau pakitus tarpinė būsena „įleidžiama“ į ciklą, kuris trunka vienu kartu mažiau, nei yra NR, taigi būtent šiam standartui ciklas suksis nuo 1 iki 9 (for round = 1 to 9). Šiame cikle tarinė būsena praeina visas keturias prieš tai išvardintas funkcijas: SubBytes, ShiftRows, MixColumns ir addRoundKey. Pasibaigus ciklui, tarpinė funkcija perduodama dar kartą praeiti pro šias funkcijas, tik šį kart nebėra funkcijos MixColumns. Kai bloko kodavimas baigtas, jis perduodamas kaip šifruotas(outByte = state).

Kiekvieną iš maišymo funkcijų peržvelgiama detaliau.

5.4.1. SubBytes

SubBytes funkcija vykdo baitų pakeitimo operaciją. Ši funkcija nuskaito tarpinės būsenos masyve esančius baitus ir sukeičia juos su baitais, esančiais S-box‘e. S-box‘as yra 16x16 matrica, kuri sudaryta sugeneruojama taip:

1. Imami visi skaičiai nuo 0 iki FF (šešioliktainė sistema) ir išreiškiami kaip GF(q^n) polinomas b(x).
2. Surandamas jam atvirkščias polinomas $b^{-1}(x)$, pasinaudojus Euklido algoritmu.
3. Apskaičiuojami S-box‘o matricos nariai pagal tokią formulę: $b_i'(x)=b_i(x)\oplus b_{(i+4)mod8}\oplus b_{(i+5)mod8}\oplus b_{(i+š)mod8}\oplus b_{(i+7)mod8}\oplus c_i$

Čia: $0<=i<8$, b_i – i-tasis polinomo narys. C_i – i-tasis baito narys iš polinomo C = 01100011 (arba {63} šešioliktinėje sistemoje). Visą šitą transformaciją paprasčiau suprasti užrašytą tokia matematikos kalba:

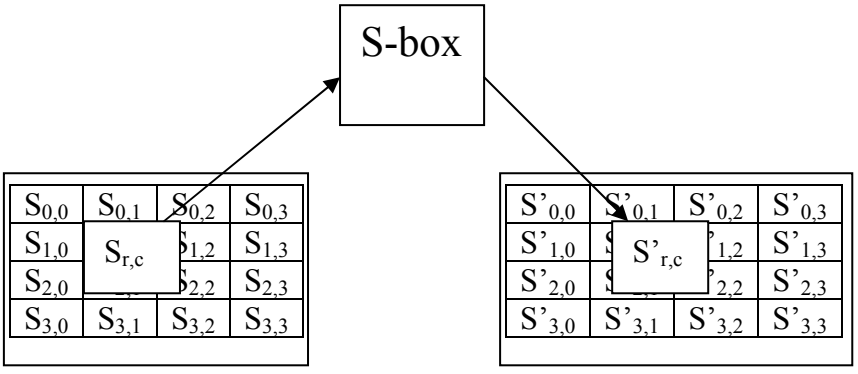
$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 10001111 \\ 11000111 \\ 11100011 \\ 11110001 \\ 11111000 \\ 01111100 \\ 00111110 \\ 00011111 \end{bmatrix} * \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Taigi sugeneravus S-box‘ą , gaunasi matrica, kuri atvaizduota 5.1 paveiksle.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
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

5. 1 pav. S-box matrica, šešioliktainėje sistemoje.

Dabar, turint tokią matricą kiekvienas tarpinės būsenos baitas, yra verčiamas į šešioliktainę sistemą ir pakeičiamas kitu skaičiumi, paimtu iš s-box matricos. Pirmasis baito šešioliktainėje sistemoje skaitmuo atitinka x eilutę, o antrasis y stulpelį iš s-box matricos ir sukeičiamas su būtent tuo nariu, kuris jį atitinka. Sekančiame paveiksle parodyta SubBytes() funkcijos schema.



5.2 pav. SubBytes funkcijos veikimo principas.

5.4.2. ShiftRows

ShiftRows funkcija pastumia tarpinės būsenos eilutės narius į kairę per tam tikrą stulpelių skaičių. Pastūmimas vyksta pagal tokią formulę:

$$S'_{r,c} = S_{r,(c + shift(r,NB))modNB}$$

Čia: $0 <= r < 4$, $0 <= c < NB$. Shift(r,NB) įgauna skirtingas reikšmes kiekvienai eilutei atskirai: Shift(0,NB) = 0, Shift(1,NB) = 1, Shift(2,NB) = 2, Shift(3,NB) = 3 (nereikia pamiršti kad AES šifravimo standarte NB = 4) . Surašius reikiamus duomenis i formulę, būtų matyti, kad pirmosios eilutės nariai lieka savo vietose, antrosios eilutės nariai yra pastumiami per 1 indeksą į kairę, trečiosios per du, o ketvirtosios per tris indeksus i kairę. Jeigu mažiausią indeksą turintį narį reikia stumti dar į kairę, jis įgauna didžiausio indekso ženklą ir nuo ten vėl stumiamas į žemesnes pozicijas reikalui esant. 5.3 paveiksle iliustruotas ShiftRows funkcijos baitų slinkimas.

S

$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$
$S_{1,0}$	$S_{1,1}$	$S_{1,2}$	$S_{1,3}$
$S_{2,0}$	$S_{2,1}$	$S_{2,2}$	$S_{2,3}$
$S_{3,0}$	$S_{3,1}$	$S_{3,2}$	$S_{3,3}$

S‘

$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$
$S_{1,1}$	$S_{1,2}$	$S_{1,3}$	$S_{1,0}$
$S_{2,2}$	$S_{2,3}$	$S_{2,0}$	$S_{2,1}$
$S_{3,3}$	$S_{3,0}$	$S_{3,1}$	$S_{3,2}$

5.3 pav. ShiftRows funkcijos veikimas.

5.4.3. MixColumns

MixColumns funkcija paima tarpinės būsenos dvimačio masyvo stulpelius, išreiškia juos kaip $GF(q^n)$ polinomą S(x) ir sudaugina su kitu polinomu $a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}x$. S‘(x) = S(x) $\oplus$ a(x) . Šito polinomo nariai kiekvienam stulpeliui atskirai yra paslenkami per vieną indeksą į dešinę. Visą šitą daugybą galima užrašyti kaip matricų sandaugą:

$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} = \begin{bmatrix} \{02\}\ \{03\}\ \{01\}\ \{01\} \\ \{01\}\ \{02\}\ \{03\}\ \{01\} \\ \{01\}\ \{01\}\ \{02\}\ \{03\} \\ \{03\}\ \{01\}\ \{01\}\ \{02\} \end{bmatrix} \begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix}$$

Taigi sustačius reikiamus duomenis gautume:

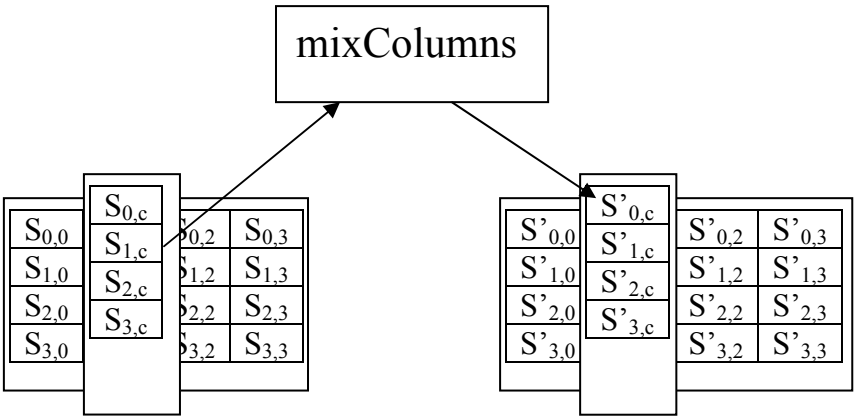
$S'_{0,c} = (\{02\} \bullet S_{0,c}) \oplus (\{03\} \bullet S_{1,c}) \oplus (\{01\} \bullet S_{2,c}) \oplus (\{01\} \bullet S_{3,c})$

$S'_{1,c} = (\{01\} \bullet S_{0,c}) \oplus (\{02\} \bullet S_{1,c}) \oplus (\{03\} \bullet S_{2,c}) \oplus (\{01\} \bullet S_{3,c})$

$S'_{2,c} = (\{01\} \bullet S_{0,c}) \oplus (\{01\} \bullet S_{1,c}) \oplus (\{02\} \bullet S_{2,c}) \oplus (\{03\} \bullet S_{3,c})$

$S'_{3,c} = (\{03\} \bullet S_{0,c}) \oplus (\{01\} \bullet S_{1,c}) \oplus (\{01\} \bullet S_{2,c}) \oplus (\{02\} \bullet S_{3,c})$

5.4-ame paveiksle pavaizduota mixColumns funkcijos transformacija tarpinei būsenai.



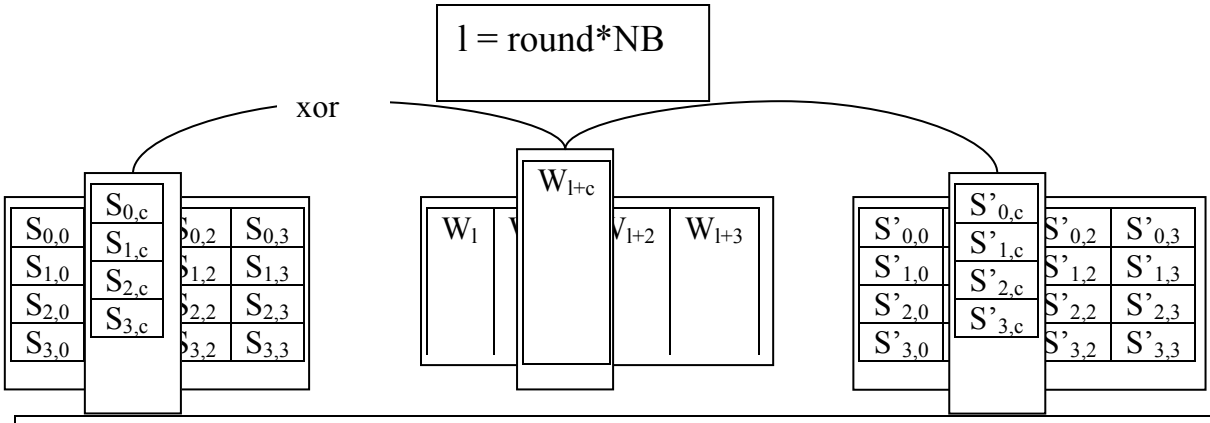
5.4 pav. MixColumns transformacija.

5.4.4. AddRoundKey

Šioje transformacijoje prie tarpinės būsenos matricos yra xor operacija pridedami sugeneruoti raktai. Visi raktai yra generuojami KeyExpansion rutinoje (aprašyta sekančiame skyriuje), kuri sugeneruoja *round**NB žodžių. Kadangi rašau programą remdamasis AES-128 standartu, tai šiuo atveju *round* = 10 , NB = 4, taigi iš viso bus sugeneruota 44 žodžiai. Šie žodžiai po 4 surašomi į dvimatį masyvą, kiekvienas jų užima 4 baitus ir sudauginami xor logine operacija su tarpinės būsenos masyvu. Visą šita supaprastintai iliustruojama taip:

$$[S'_{0,c}, S'_{1,c}, S'_{2,c}, S'_{3,c}] = [S_{0,c}, S_{1,c}, S_{2,c}, S_{3,c}] \text{ xor } [W_{round*NB+c}]$$

Kadangi informacijos bloko maišymas šiame standarte vyksta 10 kartų, tai kiekviename maišymo etape bus sugeneruojami nauji žodžiai. Taigi round nusako kelinto maišymo etapo raktai bus naudojami (imama iškart po 4 raktus kaip masyvas), o c ($0 <= c < 3$) kelinta žodį reikės naudoti. Kiekvienam stulpeliui po atskirą žodį. Ši operacija iliustruota 5.5 paveiksle:



5.5 pav. addRoundKey funkcija operuoja su kiekvienu tarpinės būsenos stulpeliu atskirai.

5.4.5. KeyExpansion rutina

KeyExpansion rutina – tai etapo raktų generavimo paprogramė. Ši paprogramė sugeneruoja round*NB žodžių (žymėsime w[i]), kurie naudojami addRoundKey funkcijoje. Taigi iš viso bus w[i] žodžių, kur 0 <= i < NB*(round + 1) . Mano atveju, round = 10, NB = 4 . Kiekvienam maišymo etapui po keturis žodžius, plius pats pirmasis etapas, dar vadinamas nuliniu. Kiekvienas žodis sudarytas iš keturių nariu užimančių po vieną baitą. Taigi žodis = 4 baitai, galime pažymėti taip: w[i] = [a_0, a_1, a_2, a_3] . Generacijai pasibaigus, reikalingi žodžiai sugrupuojami po keturis ir sudaromas 4x4 dvimatis masyvas, kuris perduodamas addRoundKey funkcijai kaip etapo raktai. Jis atrodytų štai taip:

w[0] w[1]w[2]w[3]

$$\begin{bmatrix} \begin{Bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{Bmatrix} & \begin{Bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{Bmatrix} & \begin{Bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{Bmatrix} & \begin{Bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{Bmatrix} \end{bmatrix}$$

AddRoundKey funkcija savo ruožtu šį masyvą sudeda logine xor operacija su tarpinės būsenos masyvu ir perduodama tolesniam apdirbimui.

Žodžių generacija susideda iš atskirų dalių, ją sudaro keletas funkcijų, taigi trumpai apie jas.

SubWord funkcija veikia panašiai kaip kad SubBytes. Ji operuoja kiekvienam baitui atskirai: perverčia jį į šešioliktainę sistemą ir pakeičia atitinkamu skaitmeniu iš S-Box matricos.

RotWord funkcija įvykdo žodžio narių perstūmimą per vieną poziciją į kairę. Jeigu stumiamo nario indeksas tampa minusinis, jis įgyja didžiausią reikšmę. Tarkim turim w[i] = [a_0, a_1, a_2, a_3] tai RotWord funkcija padarys: w[i] = [a_1, a_2, a_3, a_0].

Sekantis raktų generavimo dalyvis yra Rcon[] masyvas. Šio masyvo reikšmė priklauso nuo etapo numerio. Bendrai jo išraišką galima pažymėti taip: Rcon[i] = [$x^{i/NK-1}$, {00},{00},{00}] . Tai reikštų kad jis susideda iš keturių narių, kurių paskutiniųjų trijų reikšmės lygios nuliui, o pirmoji priklauso nuo etapo skaičiaus. Šiuo atveju x = 2 ir jis keliamas i/NK-1 laipsniu (i = (round+1)*NB), tačiau reikia prisiminti, kad jeigu baito reikšmė viršytų 2^7 , tuomet ji būtų sumažinta pasinaudojus moduliacijos metodu iš pirminio polinomo, kurį minėjau matematinės operacijos skirsnyje. Žinoma Rcon reikšmė kinta tik tuomet kai i/NK skaičius yra natūralusis. Žodžių generavimo cikle, Rcon[] masyvas dalyvauja sudedant logine operacija xor sugeneruotus žodžius su Rcon[] masyvo reikšme. Keletas Rcon[i] reikšmių: Rcon[0] = {00}{00}{00}{00}, Rcon[4] = {01}{00}{00}{00}, Rcon[8]={02}{00}{00}{00}, Rcon[32] = {80}{00}{00}{00}, Rcon[36]={1B}{00}{00}{00}.

Taigi aukščiau išvardinau visas funkcijas, kurios dalyvauja raktų generavime, o jų eiga yra sekančioje schemoje atvaizduotas ciklas:

```
KeyExpansion( cipherKey K[4*NB], round NR)
Dim temp as string
For i = 0 to 3
w[i] = K[(i+1)*4]
next
nk = 4
nb = 4
nr = 10
i = NK
While (i < nb * (nr + 1))
        temp = w(i - 1)
         If i Mod nk = 0 Then
           temp = SubWord(RotWord(temp)) Xor Rcon(i)
        End If
        w(i) = temp Xor w(i - nk)
        i = i + 1
    End While
```

Trumpas ciklo apibūdinimas:

Pirmieji keturi žodžiai: w[0], w[1], w[2] ir w[3] yra sudaromi tiesiog iš pagrindinio slaptažodžio. Šis turėtų būti ne trumpesnis nei 16 simbolių. Kiekvienam žodžiui yra „atriekiama“ po keturis slaptažodžio simbolius, jų kodas ASCII kodavimo sistemoje perverčiamas į šešioliktainę

sistemą ir priskiriamas pirmiesiems keturiems žodžiams. Po to vykdomas ciklas nuo i reikšmės 4 iki 44 (nb*(nr+1) = 44). Kintamąjam „*temp*“ priskiriama indeksu mažesnio žodžio reikšmė. Kiekvienam etapui generuojami keturi, žodžiai, todėl eilute „*If i Mod nk = 0 Then*“ tikrinama ar jau prasidėjo kitas etapas. Jei jis prasidėjęs, *temp* reikšmė perduodama funkcijai RotWord, o kartu ir SubWord ir po to daroma xor operacija su masyvu Rcon[i] . Tuomet naujas žodis w[i] gaunamas darant xor operacija tarp *temp* reikšmės ir keturiais indeksais žemesnio žodžio „*w(i) = temp Xor w(i - nk)*“ . Jeigu i mod NK nelygu nuliui, reiškia naujas etapas neprasidėjęs ir *temp* reikšmė neperduodama maišymo funkcijoms, o naujas žodis sudaromas tiesiog sudedant logine xor operacija žodžius *temp* ir w[i-4] (temp = w[i-1]) (pilnas raktų generacijos pavyzdys pateiktas priede 8.1).

Pastaba. Reiktų paminėti, kad AES-192 standartui iš slaptažodžio sudaromi pirmi šeši raktai, o AES-256 – aštuoni raktai. Dargi, AES-256 raktu generacija vyksta šiek tiek kitaip nei kitų dviejų standartų. Čia kiekvieno žodžių generacijos ciklo viduryje papildomai daromos SubWord modifikacijos.

5.5. DEŠIFRAVIMO ALGORITMAS

Dešifravimo algoritmas atlieka atvirkščią darbą, kuri padaro šifravimo algoritmas. Jis paima užšifruotus duomenis dvimačio masyvo pavidalu ir perleidžia juos per atitinkamas funkcijas: InvMixColumns(), InvSubBytes(), InvShiftRows() ir AddRoundKey(). Sekančiančioje schemoje parodytas dešifravimo algoritmas bendrame pavidale:

```
decrypt(inByte as array)
dim state as array = inByte
Dim round As Integer = 10
        addRoundKey(state, round)
        invShiftRows(state)
        invSubBytes(state)
        For round = 9 To 1 Step -1
           addRoundKey(state, round)
           InvMixColumns(state)
           invShiftRows(state)
           invSubBytes(state)
```

Next

Round = 0

addRoundKey(state, round)

outByte = state

Taigi prasidėjus dešifravimo procesui, jis pasiima iš kintamojo *inByte* duomenis, kurie yra dvimačio, 4x4 masyvo pavidale, priskiria juos tarpinei būsenai ir vykdo maišymo (atmaišymo) operacijas priešinga tvarka, nei kad vykdė šifravimo algoritmas. Iš pradžių tarpinė būsena perduodama addRoundKey() funkcijai su parametru round = 10, po to invShiftRows() ir invSubBytes() ir tada įleidžia į ciklą, kuris sukasi mažėjimo tvarka nuo round = 9 iki 1. Pasibaigus šiam ciklui, tarpinės būsenos duomenys dar kartą perduodami funkcijai addRoundKey() ir priskiriama kintamajam outByte – tai jau būna dešifruoti duomenys. Kiekviena iš šių funkcijų aprašyta detaliau.

5.5.1. invShiftRows

InvShiftRows funkcija atlieka atvirkščią darbą nei kad atlko ShiftRows. Ji pasiima tarpinės būsenos duomenis dvimačio masyvo 4x4 pavidale ir atlieka kiekvienos eilutės narių perstūmimą į kairę tokiu principu:

$S'_{r,(c + shift(r,NB))mod\ NB} = S_{r,c}$ čia: $0 <= r < 4$; $0 <= c < NB$ (NB = 4).

Shift(r,NB) įgyja tokias pačias reikšmes, kurios buvo paminėtos ShiftRows skyriuje, t.y.:

Shift(0,4) = 0 ; Shift(1,4) = 1 ; Shift(2,4) = 2; Shift(3,4) = 3 .

S

$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$
$S_{1,0}$	$S_{1,1}$	$S_{1,2}$	$S_{1,3}$
$S_{2,0}$	$S_{2,1}$	$S_{2,2}$	$S_{2,3}$
$S_{3,0}$	$S_{3,1}$	$S_{3,2}$	$S_{3,3}$

S‘

$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$
$S_{1,3}$	$S_{1,0}$	$S_{1,1}$	$S_{1,2}$
$S_{2,2}$	$S_{2,3}$	$S_{2,0}$	$S_{2,1}$
$S_{3,1}$	$S_{3,2}$	$S_{3,3}$	$S_{3,0}$

5.6 pav. InvShiftRows funkcijos veikimo principas.

Rezultate gaunasi, kad pirmoji tarpinės būsenos eilutės nariai lieka savo vietoje, antrosios eilutės nariai yra paslenkami per vieną narį į dešinę, trečiosios per du narius į dešinę, o ketvirtosios per tris.

5.5.2. invSubBytes

InvSubByte funkcija atkeičia tarpinės būsenos narius į tuos kurie buvo prieš įvykdant SubByte funkcija. Čia vėlgi iš GF(q^n) polinomo yra sugeneruojamas invS-box matrica, iš kurios yra suieškomi narių atitikmenys pagal šešioliktainės skaičiavimo sistemos pirmąjį ir antrąjį skaitmenis. Pirmasis atitiks x ašį, antrasis y. InvS-box matrica atvaizduota 5.7 paveiksle.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
2	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
3	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
5	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
6	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
7	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
8	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
9	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
A	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
B	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
C	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
D	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
E	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61
F	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

5.7 pav. invS-box matrica, šešioliktainėje sistemoje.

5.5.3. invMixColumns

InvMixColumns funkcija yra priešingybė MixColumns funkcijai. Ji pakeičia tarpinės būsenos masyvo narius, padaugindama juos iš tam tikros matricos. Daugyba traktuojama kaip

GF(q^n) polinomų daugyba, žymima ● simboliu, kuri buvo aprašyta matematikos veiksmų skyriuje. Sekančiai išreikšta matematikos kalba atliekamas invMixColumns funkcijos pakeitimas:

$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} = \begin{bmatrix} \{0E\}\ \{0B\}\ \{0D\}\ \{09\} \\ \{09\}\ \{0E\}\ \{0B\}\ \{0D\} \\ \{0D\}\ \{09\}\ \{0E\}\ \{0B\} \\ \{0B\}\ \{0D\}\ \{09\}\ \{0E\} \end{bmatrix} * \begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix}$$

Sudauginus gautume:

$S'_{0,c} = (\{0E\} \bullet S_{0,c}) \text{ xor } (\{0B\} \bullet S_{1,c}) \text{ xor } (\{0D\} \bullet S_{2,c}) \text{ xor } (\{09\} \bullet S_{3,c})$

$S'_{1,c} = (\{09\} \bullet S_{0,c}) \text{ xor } (\{0E\} \bullet S_{1,c}) \text{ xor } (\{0B\} \bullet S_{2,c}) \text{ xor } (\{0D\} \bullet S_{3,c})$

$S'_{2,c} = (\{0D\} \bullet S_{0,c}) \text{ xor } (\{09\} \bullet S_{1,c}) \text{ xor } (\{0E\} \bullet S_{2,c}) \text{ xor } (\{0B\} \bullet S_{3,c})$

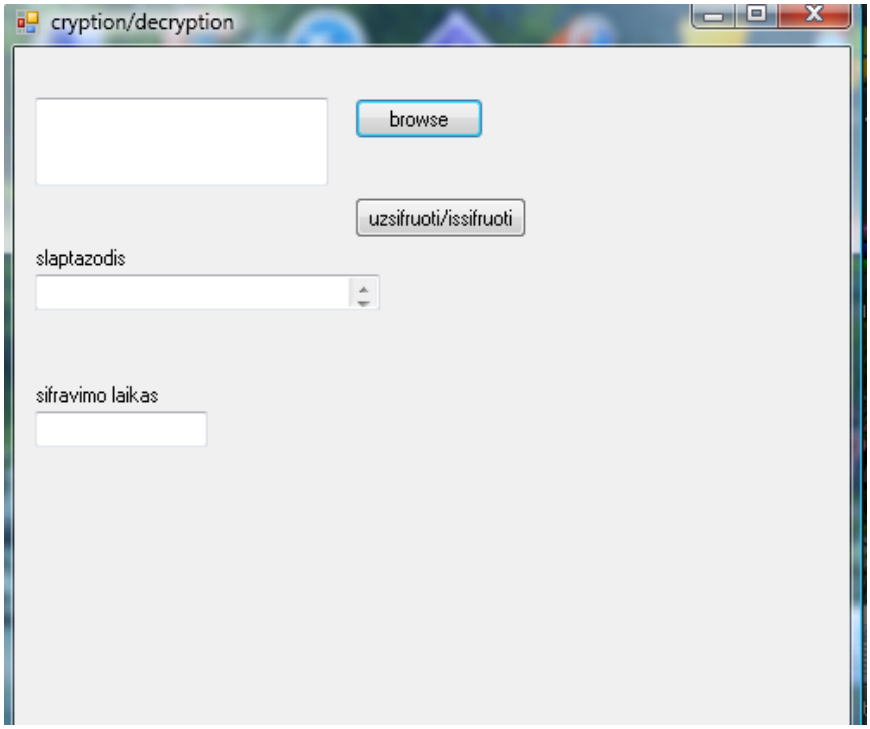
$S'_{3,c} = (\{0B\} \bullet S_{0,c}) \text{ xor } (\{0D\} \bullet S_{1,c}) \text{ xor } (\{09\} \bullet S_{2,c}) \text{ xor } (\{0E\} \bullet S_{3,c})$

6. EKSPERIMENTINĖ DALIS

6.1. AES šifravimo programa

Ankstesniuose skyriuose buvo peržvelgti skaitmeninių duomenų šifravimo metodai. Vienas patikimiausių laikomas AES standartas, todėl pagal jį, šiame darbe sukurta programinė įranga. Šiame skyriuje apžvelgiami jos parametrai.

AES programos paleidimo langas parodytas 6.1 paveiksle. Paspaudus mygtuką *browse* iššoka Windows Explorer langas ir jame naršant reikia pažymėti failą, kurį norima užšifruoti/iššifruoti. Pirmajame tuščiame teksto laukelyje automatiškai įsirašo takelis į nurodytąjį failą. Sekančiame teksto laukelyje, virš kurio yra užrašas *slaptažodis* , įvedamas unikalus slaptažodis, pagal kurį bus šifruojamas failas. Ši AES programa yra parašyta pagal AES-128 standartą ir joje naudojamas slaptažodžio ilgis turi būti 128 bitų arba 16 baitų ilgio. Kitaip tariant, programa naudoja slaptažodį sudarytą iš 16 simbolių. Šiuo atveju nebus klaidos jeigu būtų įvestas trumpesnis arba ilgesnis slaptažodis nei iš 16 simbolių, nes programa nekreips dėmesio į slaptažodžio narius, kurie yra virš 16 –ojo nario ir pridės automatiškai tiek simbolių, kiek truks iki pilno 16 baitų rakto.



6.1 pav. AES programos paleidžiamasis langas.

Suvedus takelį į norimą šifruoti failą ir įvedus slaptažodį, spaudžiamas mygtukas užšifruoti/dešifruoti. Šifruojant, programa perrenka kiekvieną failo baitą ir sumaišius juos pagal tam tikrus maišos algoritmus sukuria naują failą, su plėtiniu *.crp . Jeigu įvedant į laukelį failą, jo plėtinys bus *.crp , ji automatiškai atpažins jį kaip jau anksčiau užšifruotą ir pradės dešifravimo procesą, jeigu plėtinys bus kitoks, AES programa šį failą užšifruos.

Teksto laukelyje pavadinimu *laikas* rodomas šifravimo/dešifravimo proceso laikas, išreikštas sekundėmis.

Programos struktūra pateikta sekančiai:

Public class cryption

Sub Browse()
Sub uzsifruoti/issifruoti()
sub crypt ()

function shiftRows() as array
function subBytes() as array
function mixColumns() as array
function addRoundKey() as array

sub decrypt ()

function InvshiftRows() as array
function InvSubBytes() as array
function InvMixColumns() as array
function addRoundKey() as array

function KeyExpansion() as array

end class

Kaip praeitame skyriuje buvo detaliai aprašytas AES šifravimo standartas(žr. 5 skyrius), taip šitoje programoje yra įdiegti visos maišymo funkcijos. Programoje yra dvi paprogramės: crypt ir

decrypt. Paprogramė *uzsifruoti/issifruoti()* apsprendžia ar tai failas yra su plėtiniu *.crp ar be jo ir atitinkamai perduoda komandų vykdymą paprogramei *crypt* (jei be plėtinio *.crp) arba *decrypt* (jei su plėtiniu *.crp). Po to kiekviena iš pastarųjų paprogramių skaido failą į 16 baitų dydžio blokus ir perduoda juos atitinkamoms maišos funkcijoms *shiftRows, subBytes, mixColumns, addRoundKey arba InvshiftRows, InvSubBytes, InvMixColumns, addRoundKey.* Paskutinė *KeyExpansion* funkcija atlieka raktų generaciją. Ji iš įvesto slaptažodžio sugeneruoja 44 raktus, reikalingus tarpiniams blokų šifravimams atlikti.

6.2. AES PARAMETRAI

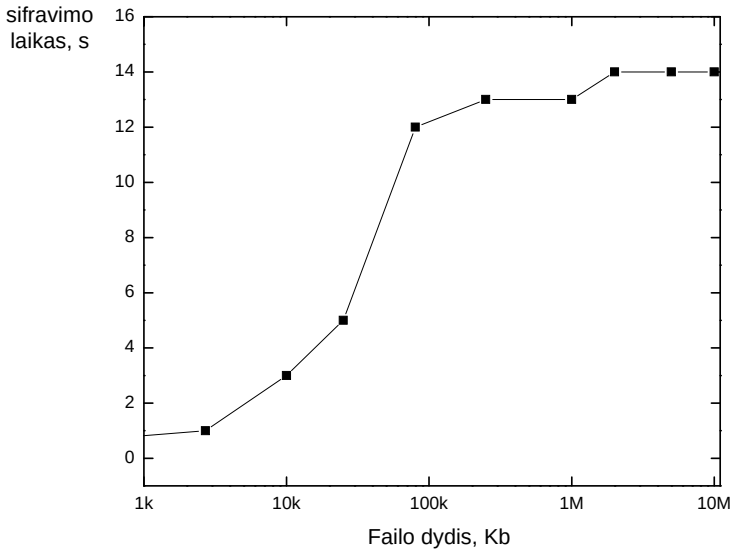
Rašant šifravimo programą, paprastesnis būdas yra pasinaudoti jau sukurtomis šifravimo klase *CryptoStream* , kuri nurodytą failą skaido į bitų seką ir perduoda šifravimo metodams, bei *AesCryptoServiceProvider* – kuri apdoroja tuos bitus pagal AES šifravimo standartą. Tačiau jau egzistuojančios klasės ir metodai visgi ilgai šifruoja didelius failus (<100 MB). Mano tikslas buvo pamėginti sukurti naują klasę ir jos metodus pagal AES-128 standartą ir palyginti jį pagal keletą parametrų su jau egzistuojančiais metodais.

6.1 lentelėje atvaizduoti įvairių dydžių failai ir jiems užšifruoti reikalingas laikas.

6.1 lentelė. Šifravimui reikalingas laikas skirtingo dydžio failams.

Failo dydis, Kb	0,032	2.7	10	25	80	250	1000	2000	5000
Šifravimo laikas, s	<1	2	3	5	12	13	13	14	14

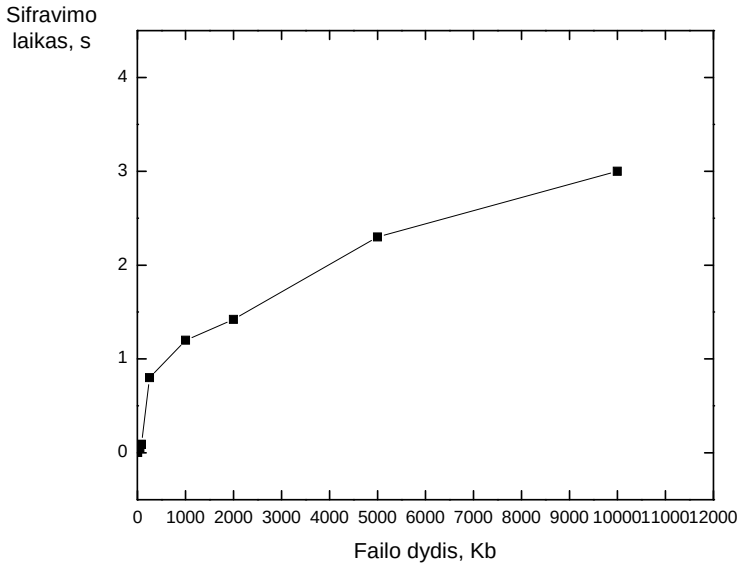
Kadangi tikslas yra sukurti programą, kuri greitai užšifruotų net ir didelius failus, buvo pasirinktas būdas šifruoti tik dalį failo. Būtent šiuo atveju, AES programa šifruoja pirmuosius 50 kilo baitų, o likusius surašo nepaliestus. Tiesioginė failų šifravimo priklausomybė nuo laiko yra atvaizduota 6.2 paveiksle.



6.2 pav. Šifravimo laiko priklausomybė nuo šifruojamo failo dydžio(naujai sukurta AES klasė).

Jeigu failas užima mažiau nei 50 kB, jis yra užšifruojamas pilnai ir laikas, reikalingas šiam procesui didėja. Jeigu failas užima daugiau nei 50 kB, kas reiškia, kad visi baitai esantys virš 50 – ojo yra tiesiog perrašomi, o šifravimo laikas nuo tam tikros vietos nebedidėja. 6.2 paveiksle matyt kad kreivė įsisotina.

Dirbant su programa, kuri įsidiegė jau egzistuojančią AES klasę ir metodus, 6.3 paveiksle aiškiai matyt, kad kreivė kyla didėjant šifruojamo failo dydžiui.



6.3 pav. Šifravimo laiko priklausomybė nuo šifruojamo failo dydžio(įdiegta AES klasė).

Taip pat buvo testuojamas sukurtos programos naudingumas, t.y. ar priklauso užšifruoto failo neatkuriamumas nuo jo dydžio ar ne ir taip pat lyginama su kitomis programomis, kuriose jau yra įdiegta standartinė AES klasė ir jos metodai. Pasirinkus keletą įvairaus dydžio *.pdf formatų failų, pasirodė naujos AES šifravimo programos trūkumai. Šifruojant failus, kurių dydis nesiekė 50KB , jie yra užšifruojami neperskaitomai, jei failai yra kelis kartus didesni, nei šifruojamas baitų kiekis, rezultatas geras, tačiau šifruojant gerokai didesnius (>10 kartų * 50 KB) failus, jie yra užkoduojami tik dalinai. Lyginant su programa, kurioje yra jau įdiegta AES klasė, pastaroji veikia be priekaištų. Šiuos parametrus iliustruoja 6.2 lentelė.

6.2 lentelė. Neperskaitoma šifruojamų failų dalis ir jų dydis KB.

pdf failo dydis, KB	< 50	68	100	250	335	4110	17000
Eksperimentinė AES klasė	100%	100%	100%	100%	22%	1,5%	<0,1%
Įdiegta AES klasė	100%	100%	100%	100%	100%	100%	100%

Dargi buvo tiriama kokiems failų formatams tokia programa tiktų, o kokiems – ne. Pasirodo, kad video ir audio formatus nepatartina šifruoti su naująją AES programa, nes kompiuteris juos netgi ir užšifruotus nesunkiai atkuria, o senieji metodai vėlgi veikia be priekaištų. Šiuos parametrus galima pamatyti 6.3 lentelėje.

6.3 lentelė. Šifruojamų failų tipai

Failo tipas	.rar	.jpg	.doc	.pdf	.mp3	.avi
Nauja AES	Veikia	Veikia	Veikia	Veikia	Neveikia	neveikia
Įdiegta AES klasė	Veikia	Veikia	Veikia	Veikia	Veikia	Veikia

7. IŠVADOS

1. Sukurtoji nauja AES šifravimo programa visiškai netinkama filmų ir muzikinių failų kodavimui.
2. Nepatikimai šifruoja didesnius nei 250 KB dydžio pdf formato failus.
3. **Labai greitai ir kokybiškai šifruoja archyvinius ir tekstinius failus**.
4. Sukurtas eksperimentinis AES šifravimo standartas Visual Basic aplinkoje neatnešė lauktų rezultatų. Šį kodą reiktų realizuoti žemesnio lygio programavimo kalba.

8. LITERATŪRA

[1]http://en.wikipedia.org/wiki/Extended_Euclidean_algorithm#Computing_a_multiplicative_inverse_in_a_finite_field

[2] Federal Information Processing Standards Publication 197 November 26, 2001 [žiūrėta 2010 balandžio 2]. Prieiga per Internetą: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[3]http://en.wikipedia.org/wiki/Advanced_Encryption_Standard

[4]http://en.wikipedia.org/wiki/Caesar_cipher

[5] http://en.wikipedia.org/wiki/Enigma_machine

[6] http://www.garykessler.net/library/crypto.html#hash

[7] http://www.progressive-coding.com/tutorial.php?id=3

[8] http://en.wikipedia.org/wiki/Modes_of_operation

[9] http://www.rsa.com/rsalabs/node.asp?id=2248

[10] http://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange

[11] Federal Information Processing Standards Publication 180-2 2002 August 1 [žiūrėta 2010 gegužės 20]. Prieiga per Internetą: http://csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice.pdf

[12] http://en.wikipedia.org/wiki/Data_Encryption_Standard

[13] FIPS PUB 46-3 FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION Reaffirmed 1999 October 25 [žiūrėta 2010 gegužės 24]. Prieiga per Internetą: http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf

[14] http://en.wikipedia.org/wiki/Extended_Euclidean_algorithm

9. TERMINŲ IR SANTRUMPŲ ŽODYNAS

9.1 TERMINAI IR AKRONIMAI

AES – (advanced encryption standart) šifravimo algoritmas.

Masyvas – kintamasis, kuriame laikomi vienas arba daugiau tokios pačios rūšies duomenys ar duomenų tipai(pvz.: sveikieji skaičiai, trupmeniniai skaičiai, simboliai ir t.t.)

Baitas – informacijos matas, kurį sudaro 8 bitai.

Bitas – informacijos matas, kuris įgyja vieną iš dviejų galimų reikšmių: vieną arba nulį.

Blokas – duomenų kiekis, su kuriuo bus atliekamos užšifravimo arba iššifravimo operacijos. Šiame aprašyme bloku galime laikyti dvimatį masyvą(susidedanti iš eilučių ir stulpelių), kurio kiekvieno nario dydis lygus 8 bitams arba vienam baitui.

Šifravimas – procesas, kuomet iš pradinio teksto, pasinaudojus įvairiomis duomenų transformacijomis ir matematinėmis operacijomis gaunamas šifruotas tekstas.

Slaptažodis – šifravimui naudojama simbolių seka. Šifravimo procese slaptažodis naudojamas Rijndael algoritme, kuriame yra sugeneruojama daug papildomų raktų, vadinamų „etapo raktais“ ir pastarieji tiesiogiai dalyvauja šifravime.

Šifruoti duomenys– tekstas arba duomenys, kurie bus naudojami dešifravime, arba gaunami pasibaigus šifravimo procesui.

Dešifravimas – procesas, kai iš šifruotų duomenų gaunami pradiniai (atvirkščia šifravimui).

Raktu generavimo rutina – procesas, kai iš slaptažodžio sugeneruojami etapo raktai.

Pradiniai duomenys– tekstas arba duomenys, kurie bus naudojami šifravime, arba gaunami pasibaigus dešifravimui.

Etapo raktai – tai šifravimo ir dešifravimo procesuose naudojami slaptažodžiai, kurie yra gaunami iš pagrindinio slaptažodžio pasinaudojus raktų generavimo rutina. Etapo raktus sudaro dvimatis masyvas sudarytas iš 4 eilučių ir NB (stulpelių skaičius, kuris priklauso nuo naudojamos AES rūšies(128, 192 ar 256 bitų atmainos))stulpelių, kiekvienas jo narys užima vieną baitą informacijos.

Tarpinė būsena – duomenys kurie dalyvauja šifravimo arba dešifravimo procesuose. Šitie duomenys yra dvimačio masyvo formoje ir susideda iš keturių eilučių ir NB(žr. NB) stulpelių.

S-box‘as – dvimatis duomenų masyvas, kuriame laikomos vieno baito dydžio skaitmenys. S-box‘as naudojamas SubBytes funkcijoje(žr. SubBytes ir InvSubBytes).

Žodis – keturių baitų dydžio duomenys. Šis terminas yra naudojamas raktų generacijoje. Etapo raktai yra sudaryti iš keturių žodžių.

9.2 PARAMETRAI IR FUNKCIJOS

a,b,c,d,e – kintamieji, skirti 32-jų bitų žodžių reikšmėms laikyti, kurie naudojami maišos reikšmėms gauti.

$H^{(i)}$ – i-oji maišos reikšmė. H^0 yra pradinė reikšmė, o H^N – rezultatas.

$H_j^{(i)}$ – j-asis žodis i-ojoje maišos reikšmėje. $H_0^{(i)}$ būtų pats kairiausias žodis, i-ojoje H reikšmėje.

K_t – konstanta, naudojama t iteracijoje.

k – skaičius, kuris nusako kiek nulių bus pridėta prie žinutės, žinutės papildymo procedūroje.

l – žinutės ilgis bitais.

m – bitų skaičius žinutės duomenų bloke $M^{(i)}$ (šiame straipsnyje jis lygus 512).

M – žinutė, kurią reikia užšifruoti.

$M^{(i)}$ – m bitų dydžio žinutės duomenų blokas.

$M_j^{(i)}$ – j-asis žodis duomenų bloke.

n – bitų skaičius, kuriuos reikės sukeisti arba perstumti žodyje, esant atitinkamai maišos funkcijai.

N – skaičius, nusakantis į kiek blokų bus suskirstyta šifruojama žinutė.

T – laikinas kintamasis, kuris įgauna w bitų reikšmę, maišos algoritmuose.

w – skaičius, nusakantis iš kiek bitų bus sudarytas žodis.

W_t – t-asis w bitų žodis maišos algoritmo procese.

AddRoundKey() – funkcija, kurios metu Tarpinė būsena sudedama XOR operacija su etapo raktais.

MixColumns() – funkcija, paimanti Tarpinės būsenos stulpelius ir pakeičia juose esančius duomenis pagal tam tikrą algoritmą.

SubBytes() - funkcija, dalyvaujanti šifravimo procese, kuri paima Tarpinės būsenos duomenis ir sukeičia juos su duomenimis iš S-box‘o, pagal tam tikra algoritmą.

ShiftRows() – funkcija, kuri paima Tarpinės būsenos eilutes ir pastumia jos narius per tam tikrą skaičių vietų į kairę.

InvMixColumns() – atvirkštinė funkcija funkcijai MixColumns. Dalyvauja dešifravimo procese.

InvSubBytes() - funkcija, atvirkštinė funkcijai SubBytes. Dalyvauja dešifravimo procese.

InvShiftRows() - funkcija, atvirkštinė funkcijai ShiftRows. Dalyvauja dešifravimo procese.

K – simbolis, naudojamas slaptažodžiui žymėti.

NB – Tarpinės būsenos stulpelių skaičius. Šiame standarte jis lygus keturiems (NB = 4).

NK – žodžių skaičius. (priklauso nuo naudojamo AES standarto:128 bitų AES NK = 4, 192 bitų NK = 6 ir 256- NK = 8).

NR – etapų skaičius parodantis kiek kartų Tarpinės būsenos duomenys bus šifruojami(priklauso nuo naudojamo AES standarto: 128 bitų AES NR = 10, 192 bitų Nr = 12 ir 256- NR = 14)

RotWord() – funkcija, naudojama raktų generavime, kuri paima sugeneruotus žodžius ir pastumia kiekviena jo narė per vieną laukelį į kairę.

SubWord() – funkcija, naudojama raktų generavime, kuri paimą žodį ir pakeičia jo narius į narius paimtus iš S-box‘o.

Rcon() – raktų generavimo etapo konstanta, kuri naudojama sekantiems etapo raktams generuoti.

9.3 ALGORITMUOSE NAUDOJAMI SIMBOLIAI

□ - loginė IR operacija.

□ - loginė ARBA operacija.

<< - bitų perkėlimo operacija. Jeigu yra x bitų žodis, tuomet n bitų yra nuimama iš kairės ir toks pats skaičius nulių prirašomas iš dešinės. Žymima $Shr^n(x)$.

>> - bitų perkėlimo operacija. Jeigu yra x bitų žodis, tai nurašoma n bitų iš dešinės ir prirašoma tiek pat nulių iš kairės. Žymima $Rotl^n(x)$.

Atvirkščia funkcija: $ROTL^n(x) \approx ROTR^{w-n}(x)$

$\oplus$ arba XOR – išskirtinė ARBA loginė operacija, kuri reikštų TAIP, jeigu tik vienas iš dviejų yra TAIP.

¬ - loginė NOT operacija.

● – simboliu žymima polinomų daugyba.

1 PRIEDAS. Raktų generacija KeyExpansion rutinoje

Norint sugeneruoti žodžius, pirmiausia reikia turėti slaptažodį K , iš kurio būtų sudaromi pirmieji keturi žodžiai. Kaip jau buvo minėta anksčiau, kiekvienas žodis susideda iš keturių baitų, taigi keturiems žodžiams reikės šešiolikos simbolių slaptažodžio. Tarkim tas slaptažodis yra: K = „MANO*Slaptazodis“ Algoritme kiekvienas slaptažodžio simbolis perverčiamas į ASCII kodą ir priskiriamas žodžiams. Šiuo atveju:

Simbolis	M	A	N	O	*	S	l	a	p	t	a	z	o	d	i	s
ASCII(dec)	77	65	78	79	42	83	108	97	112	116	97	122	111	100	105	115
ASCII(hex)	4D	41	4E	4F	2A	53	6C	61	70	74	61	7A	6F	64	69	73

O pirmieji trys žodžiai bus:

W(0) = 4D 41 4E 4F

W(1) = 2A 53 6C 61

W(2) = 70 74 61 7A

W(3) = 6F 64 69 73

Reikia nepamiršti, kad kiekvienas žodis traktuojamas kaip vienmatis masyvas susidedantis iš keturių narių w(i) = $\{a_0\ a_1\ a_2\ a_3\}$, taipgi, jų reikšmės yra šešioliktainės skaičiavimo sistemos pavidale.

Būtent mano programoje, pagrindinis slaptažodis nebūtinai turi būti sudarytas iš lygiai 16-os skaitmenų. Programa nuskaito įvestą slaptažodį, jeigu jis trumpesnis, tuomet automatiškai priekyje to slaptažodžio prirašoma tiek nuliu („0“) kiek trūksta iki šešiolikos skaitmenų, o jeigu slaptažodis ilgesnis, likę nariai tiesiog ignoruojami. Kad padidint šifravimo atsparumą, įvedžiau programoje dar papildoma slaptažodį, kurio kiekvieno simbolio šešioliktainė reikšmė pridedama xor logine operacija prie įvesto slaptažodžio simbolių reikšmių.

Taigi jau turint pirmuosius keturis slaptažodžius galima generuoti ir visus likusius:

i	temp = w(i-1)	Po funkcijos RotWord()	Po funkcijos SubWord	Rcon [i/NK]	Reikšmė po xor su Rcon[i/NK]	W(i-NK)	W(i) = temp xor W(i–NK)
4	6F646973	6469736F	43F98FA8	1000000	42F98FA8	4D414E4F	FB8C1E7
5	0FB8C1E7					2A536C61	25EBAD86

6	25EBAD86					7074617A	559FCCFC
7	559FCCFC					6F646973	3AFBA58F
8	3AFBA58F	FBA58F3A	F067380	2000000	D067380	FB8C1E7	2BEB267
9	2BEB267					25EBAD86	27551FE1
10	27551FE1					559FCCFC	72CAD31D
11	72CAD31D					3AFBA58F	48317692
12	48317692	31769248	C7384F52	4000000	C3384F52	2BEB267	C186FD35
13	C186FD35					27551FE1	E6D3E2D4
14	E6D3E2D4					72CAD31D	941931C9
15	941931C9					48317692	DC28475B
16	DC28475B	28475BDC	34A03986	8000000	3CA03986	C186FD35	FD26C4B3
17	FD26C4B3					E6D3E2D4	1BF52667
18	1BF52667					941931C9	8FEC17AE
19	8FEC17AE					DC28475B	53C450F5
20	53C450F5	C450F553	1C53E6ED	10000000	C53E6ED	FD26C4B3	F175225E
21	F175225E					1BF52667	EA800439
22	EA800439					8FEC17AE	656C1397
23	656C1397					53C450F5	36A84362
2	36A84362	A8436236	C21AAA0	20000000	E21AAA0	F175225E	136F885B

4			5		5		
25	136F885B					EA800439	F9EF8C62
26	F9EF8C62					656C1397	9C839FF5
27	9C839FF5					36A84362	AA2BDC97
28	AA2BDC97	2BDC97AA	F18688AC	40000000	B18688AC	136F885B	A2E900F7
29	A2E900F7					F9EF8C62	5B068C95
30	5B068C95					9C839FF5	C7851360
31	C7851360					AA2BDC97	6DAECFF7
32	6DAECFF7	AECFF76D	E48A683C	80000000	648A683C	A2E900F7	C66368CB
33	C66368CB					5B068C95	9D65E45E
34	9D65E45E					C7851360	5AE0F73E
35	5AE0F73E					6DAECFF7	374E38C9
36	374E38C9	4E38C937	2F07DD9A	1B000000	3407DD9A	C66368CB	F264B551
37	F264B551					9D65E45E	6F01510F
38	6F01510F					5AE0F73E	35E1A631
39	35E1A631					374E38C9	2AF9EF8
40	2AF9EF8	AF9EF802	790B4177	36000000	4F0B4177	F264B551	BD6FF426

41	BD6FF426					6F01510F	D26EA529
42	D26EA529					35E1A631	E78F0318
43	E78F0318					2AF9EF8	E5209DE0

Taip atrodo visi sugeneruoti 44 žodžiai. Vykdant addRoundKey funkcija, žodžiai grupuojami po keturis į dvimatį masyvą, ir sudedami logine xor operacija su tarpinės būsenos masyvu. Tarkim: esant nuliniam etapui addRoundKey funkcija pasiims pirmus keturis žodžius kaip masyvą, kuris atrodys taip:

$$\begin{matrix} W0 & W1 & W2 & W3 \end{matrix}$$
$$\begin{bmatrix} 4D & 2A & 70 & 6F \\ 41 & 53 & 74 & 64 \\ 4E & 6C & 61 & 69 \\ 4F & 61 & 7A & 73 \end{bmatrix}$$

2 PRIEDAS. AES šifravimo pavyzdys

KeyExpansion skyriuje parodyta kaip yra sugeneruojami raktai, šiame skyriuje pavaizduojamas pavyzdya, kaip yra sumaišomas duomenų blokas panaudojus visus tuos raktus ir kitas šifravimo procese dalyvaujančias funkcijas.

Tarkim turime failą, kurį reikia užšifruoti. Pasinaudodami tam tikrais klasių metodais, mes tą failą nuskaitom po baitą, suskirstom grupėmis po šešiolika baitų ir surašom į dvimačius masyvus. Tuomet kiekviena tokį šešiolikos baitų masyvą perleidžiame per šifravimo ciklą ir surašom į kitą kintamąjį. Po to paimam kitą bloką, jį vėl užkoduojam ir pridedam prie jau ankstesnių duomenų į tą patį kintamąjį. Ir taip procedūrą kartojam tol, kol visi failo duomenys yra užšifruojami ir surašomi į kažkokį tai kintamąjį. Tada sukuriamas naujas failas ir visi tame kintamajame esantys sumaišyti baitai surašomi į tą naują failą. Naujasis failas gali turėti specialų plėtinį, tarkim *.crp . Senąjį failą (tą kurio duomenis šifravome) galime ištrinti. Taigi sekančiame pavyzdyje parodyta kaip vienas iš tokių galimų duomenų blokų yra žingsnis po žingsnio užkoduojamas.

Tarkim nuskaitėm pirmuosius failo, kurį norime užšifruoti, šešiolika baitų, juos pervertėme į šešioliktainę sistemą ir surašėme i dvimatį masyvą. Taigi turim :

Slaptažodis = „MANO*Slaptazodis“

K = 4D 41 4E 4F 2A 53 6C 61 70 74 61 7A 6F 64 69 73

Nuskaityti pirmieji 16 baitų = 32 43 F6 A8 88 5A 30 8D 31 31 98 A2 E0 37 07 34

Surašyta į bloką:

inByte =

32	88	31	E0
43	5A	31	37
F6	30	98	07
A8	8D	A2	34

Etapo pradžia	Po SubBytes	Po shiftRows	Po MixColumn	⊕	Etapo raktas
32 88 31 E0 43 5A 31 37 F6 30 98 07 A8 8D A2 34					4D 2A 70 6F 41 53 74 64 4E 6C 61 69 4F 61 7A 73
7F A2 41 8F 2 9 45 53 B8 5C F9 6E E7 EC D8 47	D2 3A 83 73 77 1 6E ED 6C 4A 99 9F 94 CE 61 A0	D2 3A 83 73 1 6E ED 77 99 9F 6C 4A A0 94 CE 61	85 CD 93 54 C0 C8 38 22 1 D6 FF 33 AE 8C 98 6A		F 25 55 3A B8 EB 9F FB C1 AD CC A5 E7 86 FC 8F
8A E8 C6 6E 78 23 A7 D9 C0 7B 33 96 49 A 64 E5	7E 9B B4 9F BC 26 5C 35 BA 21 C3 90 3B 67 43 D9	7E 9B B4 9F 26 5C 35 BC C3 90 BA 21 D9 3B 67 43	8C 62 F1 98 B5 B3 6C DC B5 B1 47 A4 CE C 86 A1		2 27 72 48 BE 55 CA 31 B2 1F D3 76 67 E1 1D 92
8E 45 83 D0 B E6 A6 ED 7 AE 94 D2 A9 ED 9B 33	19 6E EC 70 2B 8E 24 55 C5 E4 22 B5 D3 55 14 C3	19 6E EC 70 8E 24 55 2B 22 B5 C5 E4 C3 D3 55 14	5A D6 AC 6D BB 31 47 5 8D 55 D7 B4 1A 9E 15 77		C1 E6 94 DC 86 D3 19 28 FD E2 31 47 35 D4 C9 5B
9B 30 38 B1 3D E2 5E 2D 70 B7 E6 F3 2F 4A DC 2C	14 4 7 C8 27 98 58 D8 51 A9 8E D 15 D6 86 71	14 4 7 C8 98 58 D8 27 8E D 51 A9 71 15 D6 86	64 F8 FA CD 18 B6 89 E0 18 79 1C 37 C8 73 37 DA		FD 1B 8F 53 26 F5 EC C4 C4 26 17 50 B3 67 AE F5
99 E3 75 9E E1 43 65 24 DC 5F B 67 7B 14 99 2F	EE 11 9D B F8 1A 4D 36 86 CF 2B 85 21 FA EE 15	EE 11 9D B 1A 4D 36 F8 2B 85 86 CF 15 21 FA EE	D7 51 7 24 B2 3E 9A 44 9D 2E A9 5F 32 B9 E3 ED		F1 EA 65 36 75 80 5C A8 22 04 13 43 5E 39 97 62
26 BB 62 12 C7 BE F6 EC BF 2A BA 1C 6C 80 74 8F	F7 EA AA C9 C6 AE 42 CE 8 E5 F4 9C 50 CD 92 73	F7 EA AA C9 AE 42 CE C6 F4 9C 8 E5 73 50 CD 92	9B C5 C3 AF C4 81 F8 F8 3F 7B 38 73 BE 5B A2 5C		13 F9 9C AA 6F EF 83 2B 88 8C 9F DC 5B 62 F5 97
88 3C 5F 5 AB 6E 7B D3 B7 F7 A7 AF E5 39 57 CB	C4 EB CF 6B 62 9F 21 66 A9 68 5C 79 D9 12 5B 1F	C4 EB CF 6B 9F 21 66 62 5C 79 A9 68 1F D9 12 5B	6A E 94 43 1A FB F1 AC C2 48 D6 34 AA D7 A1 1		A2 5B C7 6D E9 06 85 AE 00 8C 13 CF F7 95 60 F7
C8 55 53 2E F3 FD 74 E2 C2 C4 C5 FB 5D 42 C1 F6	E8 FC ED 31 D 54 92 98 25 1C A6 F 4C 2C 78 42	E8 FC ED 31 54 92 98 D A6 F 25 1C 42 4C 2C 78	D3 D 7B 11 F3 9E 85 77 2D A4 4B 8C 55 1A C9 B2		C6 9D 5A 37 63 65 E0 4E 68 E4 F7 38 CB 5E 3E C9
15 90 21 26 90 FB 65 39 45 40 BC B4 9E 44 F7 7B	59 60 FD F7 60 F 4D 12 6E 9 65 8D B 1B 68 21	59 60 FD F7 F 4D 12 60 65 8D 6E 9 21 B 1B 68	E7 91 A2 34 C9 7D 70 44 FF 31 1E 3D C3 76 56 BB		F2 6F 35 2 64 01 E1 AF B5 51 A6 9E 51 0F 31 F8
15 FE 97 36 AD 7C 91 EB 4A 60 B8 A3 92 79 67 43	59 BB 88 5 95 10 81 E9 D6 D0 6C A 4F B6 85 1A	59 BB 88 5 10 81 E9 95 6C A D6 D0 1A 4F B6 85			BD D2 E7 E5 6F 6E 8F 20 F4 A5 03 9D 26 29 18 E0

OutByte =

E4	69	6F	E0
7F	EF	66	B5
98	AF	D5	4D
3C	66	AE	65

Taigi pirmieji 128 duomenų bitai būtų sumaišyti, sukratyti ir perversti šitokia seka.

3 PRIEDAS. SHA-1 šifravimo pavyzdys

Turim žinutę M = „abc“. Ją reikia užšifruoti pagal SHA-1 algoritmą. Kiekvieną simbolį perverčiam į ASCII kodą dvejetainėje sistemoje ir pridedam papildymą. Gaunasi lygiai vienas 512 bitų blokas :

$$M = \overbrace{01100001}^{"a"}\overbrace{01100010}^{"b"}\overbrace{01100011}^{"c"}\overbrace{1000000.....000000000}^{k=423}\overbrace{...011000}^{64 bitai}$$

Priskiriam reikšmes pirmiesiems H:

$H_0^{(0)}$ = 67452301

$H_1^{(0)}$ = efcdab89

$H_2^{(0)}$ = 98badcfe

$H_3^{(0)}$ = 10325476

$H_4^{(0)}$ = c3d2e1f0.

Priskiriamos reikšmės žodžiams W (bitai iš duomenų bloko perverčiami į šešioliktainę sistemą):

W_0 = 61626380 W_1 = 00000000 W_2= 00000000 W_3 = 00000000

W_4 = 00000000 W_5 = 00000000 W_6 = 00000000 W_7 = 00000000

W_8 = 00000000 W_9 = 00000000 W_{10} = 00000000 W_{11} = 00000000

W_{12} = 00000000 W_{13} = 00000000 W_{14} = 00000000 W_{15} = 00000018

Visi likę žodžiai W_t 16<= t <= 79 turi reikšmes 00000000.

Generuojamos reikšmės kintamiesiems a,b,c,d,e kiekvieno ciklo (nuo t = 0 iki 79) etape:

t	a	b	c	d	e
0	0116fc33	67452301	7bf36ae2	98badcfe	10325476
1	8990536d	0116fc33	59d148c0	7bf36ae2	98badcfe
2	a1390f08	8990536d	c045bf0c	59d148c0	7bf36ae2
3	cdd8e11b	a1390f08	626414db	c045bf0c	59d148c0
4	cfd499de	cdd8e11b	284e43c2	626414db	c045bf0c
5	3fc7ca40	cfd499de	f3763846	284e43c2	626414db
6	993e30c1	3fc7ca40	b3f52677	f3763846	284e43c2

7	9e8c07d4	993e30c1	0ff1f290	b3f52677	f3763846
8	4b6ae328	9e8c07d4	664f8c30	0ff1f290	b3f52677
9	8351f929	4b6ae328	27a301f5	664f8c30	0ff1f290
10	fbda9e89	8351f929	12dab8ca	27a301f5	664f8c30
11	63188fe4	fbda9e89	60d47e4a	12dab8ca	27a301f5
12	4607b664	63188fe4	7ef6a7a2	60d47e4a	12dab8ca
13	9128f695	4607b664	18c623f9	7ef6a7a2	60d47e4a
14	196bee77	9128f695	1181ed99	18c623f9	7ef6a7a2
15	20bdd62f	196bee77	644a3da5	1181ed99	18c623f9
16	4e925823	20bdd62f	c65afb9d	644a3da5	1181ed99
17	82aa6728	4e925823	c82f758b	c65afb9d	644a3da5
18	dc64901d	82aa6728	d3a49608	c82f758b	c65afb9d
19	fd9e1d7d	dc64901d	20aa99ca	d3a49608	c82f758b
20	1a37b0ca	fd9e1d7d	77192407	20aa99ca	d3a49608
21	33a23bfc	1a37b0ca	7f67875f	77192407	20aa99ca
22	21283486	33a23bfc	868dec32	7f67875f	77192407
23	d541f12d	21283486	0ce88eff	868dec32	7f67875f
24	c7567dc6	d541f12d	884a0d21	0ce88eff	868dec32
25	48413ba4	c7567dc6	75507c4b	884a0d21	0ce88eff
26	be35fbd5	48413ba4	b1d59f71	75507c4b	884a0d21
27	4aa84d97	be35fbd5	12104ee9	b1d59f71	75507c4b
28	8370b52e	4aa84d97	6f8d7ef5	12104ee9	b1d59f71
29	c5fbaf5d	8370b52e	d2aa1365	6f8d7ef5	12104ee9
30	1267b407	c5fbaf5d	a0dc2d4b	d2aa1365	6f8d7ef5
31	3b845d33	1267b407	717eebd7	a0dc2d4b	d2aa1365
32	046faa0a	3b845d33	c499ed01	717eebd7	a0dc2d4b
33	2c0ebc11	046faa0a	cee1174c	c499ed01	717eebd7
34	21796ad4	2c0ebc11	811bea82	cee1174c	c499ed01
35	dcbbb0cb	21796ad4	4b03af04	811bea82	cee1174c
36	0f511fd8	dcbbb0cb	085e5ab5	4b03af04	811bea82
37	dc63973f	0f511fd8	f72eec32	085e5ab5	4b03af04
38	4c986405	dc63973f	03d447f6	f72eec32	085e5ab5
39	32de1cba	4c986405	f718e5cf	03d447f6	f72eec32

40	fc87dedf	32de1cba	53261901	f718e5cf	03d447f6
41	970a0d5c	fc87dedf	8cb7872e	53261901	f718e5cf
42	7f193dc5	970a0d5c	ff21f7b7	8cb7872e	53261901
43	ee1b1aaf	7f193dc5	25c28357	ff21f7b7	8cb7872e
44	40f28e09	ee1b1aaf	5fc64f71	25c28357	ff21f7b7
45	1c51e1f2	40f28e09	fb86c6ab	5fc64f71	25c28357
46	a01b846c	1c51e1f2	503ca382	fb86c6ab	5fc64f71
47	bead02ca	a01b846c	8714787c	503ca382	fb86c6ab
48	baf39337	bead02ca	2806e11b	8714787c	503ca382
49	120731c5	baf39337	afab40b2	2806e11b	8714787c
50	641db2ce	120731c5	eebce4cd	afab40b2	2806e11b
51	3847ad66	641db2ce	4481cc71	eebce4cd	afab40b2
52	e490436d	3847ad66	99076cb3	4481cc71	eebce4cd
53	27e9f1d8	e490436d	8e11eb59	99076cb3	4481cc71
54	7b71f76d	27e9f1d8	792410db	8e11eb59	99076cb3
55	5e6456af	7b71f76d	09fa7c76	792410db	8e11eb59
56	c846093f	5e6456af	5edc7ddb	09fa7c76	792410db
57	d262ff50	c846093f	d79915ab	5edc7ddb	09fa7c76
58	09d785fd	d262ff50	f211824f	d79915ab	5edc7ddb
59	3f52de5a	09d785fd	3498bfd4	f211824f	d79915ab
60	d756c147	3f52de5a	4275e17f	3498bfd4	f211824f
61	548c9cb2	d756c147	8fd4b796	4275e17f	3498bfd4
62	b66c020b	548c9cb2	f5d5b051	8fd4b796	4275e17f
63	6b61c9e1	b66c020b	9523272c	f5d5b051	8fd4b796
64	19dfa7ac	6b61c9e1	ed9b0082	9523272c	f5d5b051
65	101655f9	19dfa7ac	5ad87278	ed9b0082	9523272c
66	0c3df2b4	101655f9	0677e9eb	5ad87278	ed9b0082
67	78dd4d2b	0c3df2b4	4405957e	0677e9eb	5ad87278
68	497093c0	78dd4d2b	030f7cad	4405957e	0677e9eb
69	3f2588c2	497093c0	de37534a	030f7cad	4405957e
70	c199f8c7	3f2588c2	125c24f0	de37534a	030f7cad
71	39859de7	c199f8c7	8fc96230	125c24f0	de37534a
72	edb42de4	39859de7	f0667e31	8fc96230	125c24f0

73	11793f6f	edb42de4	ce616779	f0667e31	8fc96230
74	5ee76897	11793f6f	3b6d0b79	ce616779	f0667e31
75	63f7dab7	5ee76897	c45e4fdb	3b6d0b79	ce616779
76	a079b7d9	63f7dab7	d7b9da25	c45e4fdb	3b6d0b79
77	860d21cc	a079b7d9	d8fdf6ad	d7b9da25	c45e4fdb
78	5738d5e1	860d21cc	681e6df6	d8fdf6ad	d7b9da25
79	42541b35	5738d5e1	21834873	681e6df6	d8fdf6ad

Taigi 79 žodžiai W yra sugeneruoti iš vienintelio žinutės M bloko. Toliau perduodamos sekančios reikšmės H kintamajam:

$H_0^{(1)}$ = 67452301 + 42541b35 = a9993e36
$H_1^{(1)}$ = efcdab89 + 5738d5e1 = 4706816a
$H_2^{(1)}$ = 98badcfe + 21834873 = ba3e2571
$H_3^{(1)}$ = 10325476 + 681e6df6 = 7850c26c
$H_4^{(1)}$ = c3d2e1f0 + d8fdf6ad = 9cd0d89d.

Rezultatas yra 160 bitų seka (perversta į šešioliktainę sistemą)
$H^N = H_0^N \square H_1^N \square H_2^N \square H_3^N \square H_4^N$ = a9993e36 4706816a ba3e2571 7850c26c 9cd0d89d.

4 PRIEDAS. Žinutės užpildymas (*message padding*).

Dar vienas svarbus niuansas naudojamas šifravimo metoduose yra taip vadinamas „žinutės užpildymas“ (*message padding*) . Esmė tokia, kad failas yra šifruojamas fiksuoto ilgio blokais (tarkim imant po 128 bitus) ir taip šifruojamas iki pabaigos. Bet visas failo dydis gali nesidalinti be liekanos iš bloko dydžio. Kitaip tariant failo dydis nesudaro pilnus n blokų. Dažniausiai šifravimo proceso metu, priartėjus prie failo galo, lieka nepilnas blokas. Tuomet yra naudojamas šis būdas tai problemai išspręsti. Principas toks, kad prie failo pradžios pridedama tiek „tuščių“ baitų, kiek reikia, kad failo dydis būtų dalus be liekanos iš šifruojamųjų blokų dydžio.

Žinutės užpildymas maišos funkcijų SHA-1 ir SHA-256 algoritmuose.

Tarkim turime žinutę M, kuri yra *l* bitų ilgio. Reikia ją užpildyti, kad jos ilgis būtų dalus iš skaičiaus 512. Tuomet prie *l* bitų pridedam vienetą (+ 1) k nulių. K yra toks skaičius, kad sekanti lygybė būtų teisinga: $l + 1 + k = 448$. Tuomet dar pridedam 64 bitų ilgio bloką, kurio skaitinė reikšmė yra lygi *l* reikšmei.

Pavyzdžiui turime žinutę „**abc**“ . Pervertus kiekvieną jos simbolį (a, b ir c) į ASCII kodą ir išreiškus per dvejetainę sistemą turime tokį rezultatą:

a = 01100001 , b = 01100010 ir c = 01100011

Dabar prie šitos sekos iš galo pridedamas 1 ir k nulių. K= 448 – l – 1 = 448 – 24 -1 = 423. Taigi pridedamas 1 ir 423 nuliai. Gavosi tokia seka:

$$\overbrace{01100001}^{"a"}\overbrace{01100010}^{"b"}\overbrace{01100011}^{"c"}\overbrace{1000000.....000000}^{k=423}$$

Dabar reikia pridėti 64 bitų bloką, kurio skaitmeninis dydis būtų lygus *l*. Taigi l = 24, tai reiškia 64 bitų seka būtų: 00000.....011000. Ir visa užpildyta žinutė:

$$\overbrace{01100001}^{"a"}\overbrace{01100010}^{"b"}\overbrace{01100011}^{"c"}\overbrace{1000000.....000000}^{k=423}\overbrace{000...011000}^{64 bitai}$$