Napredni standard šifriranja

Vuković, Filip

Sveučilište J. J. Strossmayera u Osijeku
Fakultet primijenjene matematike i informatike
Sveučilišni diplomski studij matematike
(smjer: Matematika i računarstvo)

Filip Vuković

Advanced encryption standard

Diplomski rad

Osijek, 2024.

Sveučilište J. J. Strossmayera u Osijeku
Fakultet primijenjene matematike i informatike
Sveučilišni diplomski studij matematike
(smjer: Matematika i računarstvo)

Filip Vuković

Advanced encryption standard

Diplomski rad

Mentor: doc. dr. sc. Domagoj Ševerdija

Komentor: dr. sc. Luka Borozan

Osijek, 2024.

Sadržaj

1 Uvod **1**
- 1.1 Definicije . . . 1
- 1.2 Pojmovi i akronimi . . . 1
- 1.3 Funkcije AES algoritma . . . 2
- 1.4 Parametri i simboli algoritma . . . 3

2 Oznake i konvencije **4**
- 2.1 Ulazi i izlazi . . . 4
- 2.2 Bajtovi . . . 4
- 2.3 Indeksiranje nizova bajtova . . . 4
- 2.4 Stanje . . . 5
- 2.5 Nizovi riječi . . . 6

3 Matematički preduvjeti **7**
- 3.1 Zbrajanje u GF(2^8) . . . 7
- 3.2 Množenje u GF(2^8) . . . 7
- 3.3 Množenje riječi s fiksnom matricom . . . 8
- 3.4 Multiplikativni inverzi u GF(2^8) . . . 9

4 Specifikacija algoritma **10**
- 4.1 Cipher() . . . 10
  - 4.1.1 SubBytes() . . . 11
  - 4.1.2 ShiftRows() . . . 13
  - 4.1.3 MixColumns() . . . 13
  - 4.1.4 AddRoundKey() . . . 14
- 4.2 KeyExpansion() . . . 14
- 4.3 InvCipher() . . . 16
  - 4.3.1 InvShiftRows() . . . 16
  - 4.3.2 InvSubBytes() . . . 17
  - 4.3.3 InvMixColumns() . . . 17
  - 4.3.4 Inverzija AddRoundKey() . . . 18

5 Razmatranja pri Implementaciji **19**
- 5.1 Key Length Requirements . . . 19
- 5.2 Keying Restrictions . . . 19
- 5.3 Parameter Extensions . . . 19
- 5.4 Implementation Suggestions Regarding Various Platforms . . . 19

6 Modovi operacija **20**
- 6.1 Padding za AES Algoritam . . . 20
- 6.2 Electronic Codebook (ECB) . . . 21
- 6.3 Cipher Block Chaining (CBC) . . . 22
- 6.4 Cipher Feedback (CFB) . . . 23
- 6.5 Output Feedback (OFB) . . . 24
- 6.6 Counter (CTR) . . . 25

7 Napadi na Modove operacija **26**
7.1 Electronic Codebook (ECB) . . . 26
7.2 Cipher Block Chaining (CBC) . . . 26
7.3 Cipher Feedback (CFB) . . . 27
7.4 Output Feedback (OFB) . . . 27
7.5 Counter (CTR) . . . 27
7.6 Padding Oracle napad na CBC mod . . . 28
7.6.1 Opis napada . . . 28
7.6.2 Praktični primjer: Šifriranje korisničkih podataka u CBC modu . . . 28
7.6.3 Kako napad funkcionira . . . 28
7.6.4 Implementacija napada . . . 28
7.7 Otkrivanje obrazaca u ECB modu . . . 29
7.7.1 Primjer ranjivosti: Otkrivanje obrazaca u šifriranim podacima . . . 29
7.7.2 Praktični primjer: Šifriranje slike u ECB modu . . . 29
7.7.3 Napad na ECB mod: Otkrivanje tajne poruke . . . 29
7.7.4 Implementacija napada . . . 30
7.8 Zaključak . . . 30

8 Primjene AES-a u stvarnom svijetu **31**
8.1 Sigurnost podataka . . . 31
8.2 Komunikacijski protokoli . . . 31
8.3 Financijske transakcije . . . 31
8.4 Zaštita intelektualnog vlasništva . . . 32
8.5 Vladine i vojne primjene . . . 32
8.6 Zaključak . . . 32

A Dodatak: popis funkcija i programa **33**
A.1 Python Implementacija Padding Oracle Napada na CBC Mod . . . 40
A.2 Python Implementacija Napada na ECB Mod . . . 42

Literatura **44**

Sažetak **45**

Abstract **46**

Životopis **47**

1 Uvod

Blok je niz bitova određene fiksne duljine. Blok šifra je obitelj permutacija blokova koja je parametrizirana nizom bitova zvanim ključ. Blok šifre su temelj za mnoge kriptografske usluge, osobito one koje osiguravaju povjerljivost podataka.

U 1997. godini, Nacionalni institut za standarde i tehnologiju (NIST) započeo je razvojni proces za Napredni standard za šifriranje (engl. Advanced Encryption Standard, u daljnjem tekstu AES) i pozvao javnost da podnese kandidatske algoritme za blok šifre. 2000. godine NIST je objavio izbor algoritma Rijndael za AES. Ovaj standard specificira tri implementacije Rijndaela: AES-128, AES-192 i AES-256, gdje sufiks označava duljinu ključa u bitovima. Duljina bloka (tj. duljina ulaznih i izlaznih podataka) je 128 bita u svakom slučaju. Rijndael podržava dodatne duljine blokova i ključeva koje nisu usvojene u ovom standardu. Rijndael algoritam je detaljno opisan u radu "Rijndael: A New Block Cipher", objavljenom 2001. godine od strane autora Daemen i Rijmen [1].

AES je simetrični ključni algoritam za šifriranje podataka kojeg je usvojio Nacionalni institut za standarde i tehnologiju (NIST) 2001. godine. Koristi blok šifru s duljinom bloka od 128 bita i ključevima duljine 128, 192 ili 256 bita.

Algoritam AES-a sastoji se od nekoliko koraka transformacije podataka, uključujući zamjenu bajtova (SubBytes), premještanje redova (ShiftRows) te miješanje stupaca (MixColumns).

U ovom radu, osim opisa teorijskih aspekata AES algoritma i načina rada, prezentirat će se i praktični aspekti kroz implementaciju u programskom jeziku C. Implementacija pokriva pet najčešćih načina rada i pruža pregled njihovih karakteristika, performansi i primjenjivosti u različitim scenarijima. Cilj ovog rada je pružiti sveobuhvatno razumijevanje AES algoritma i njegovih načina rada te demonstrirati praktičnu primjenu kroz implementaciju u C-u.

1.1 Definicije

1.2 Pojmovi i akronimi

Sljedeće definicije se koriste u ovom standardu:

AES Napredni standard za šifriranje.

Niz (Array) Struktura podataka fiksne veličine koja pohranjuje zbirku elemenata, gdje je svaki element identificiran svojim cijelim brojem (indeksom ili indeksima).

Bit Binarna znamenka: 0 ili 1.

Blok Niz bitova određene fiksne duljine. U ovom standardu, blokovi se sastoje od 128 bitova, ponekad prikazanih kao nizovi bajtova ili riječi.

Blok šifra Obitelj permutacija blokova koja je parametizirana ključem.

Bajt (Byte) Niz od osam bitova.

Ključ (Key) Parametar blok šifre koji određuje izbor permutacije iz obitelji blok šifri.

Key schedule Niz ključeva rundi koji su generirani iz ključa pomoću funkcije KeyExpansion().

Rijndael Blok šifra koju je NIST odabrao kao pobjednika AES natjecanja.

Runda (Round) Niz transformacija stanja koji se ponavlja Nr puta u specifikacijama funkcija Cipher() i InvCipher(). Niz se sastoji od četiri transformacije, osim u jednoj iteraciji, u kojoj se jedna od transformacija izostavlja.

Ključ runde (Round key) Jedan od Nr + 1 nizova od četiri riječi koji su izvedeni iz ključa blok šifre pomoću rutine za proširenje ključeva; svaki ključ runde je ulaz za instancu funkcije AddRoundKey() u AES blok šifri.

Stanje (State) Međurezultat AES blok šifre koji je predstavljen kao dvodimenzionalni niz bajtova s četiri retka i četiri stupca.

S-tablica (S-box) Nelinearna zamjenska tablica koja se koristi u funkcijama SubBytes() i KeyExpansion() za izvedbu jedan-na-jedan zamjene vrijednosti bajta.

Riječ (Word) Grupa od 32 bita koja se tretira kao jedinstvena cjelina ili kao niz od 4 bajta.

1.3 Funkcije AES algoritma

Sljedeće funkcije su specificirane u ovom standardu:

AddRoundKey() Transformacija stanja u kojoj se ključ runde kombinira sa stanjem.

AES-128() Blok šifra specificirana u ovom standardu s 128-bitnim ključevima.

AES-192() Blok šifra specificirana u ovom standardu s 192-bitnim ključevima.

AES-256() Blok šifra specificirana u ovom standardu s 256-bitnim ključevima.

Cipher() Transformacija blokova koja je osnova za AES-128, AES-192 i AES-256; raspored ključeva i broj rundi su parametri transformacije.

InvCipher() Inverz funkcije Cipher().

InvMixColumns() Inverz funkcije MixColumns().

InvSBOX() Inverz funkcije SBOX().

InvShiftRows() Inverz funkcije SHIFTROWS().

InvSubBytes() Inverz funkcije SubBytes().

KeyExpansion() Rutina koja generira ključeve runde iz ključa.

MixColumn() Transformacija stanja koja uzima sve stupce stanja i miješa njihove podatke (neovisno jedne o drugima) kako bi se proizveli novi stupci.

RotWord() Transformacija riječi u kojoj se četiri bajta riječi ciklički permutiraju.

SBOX() Transformacija bajtova definirana S-kutijom.

ShiftRows() Transformacija stanja u kojoj se posljednja tri retka ciklički pomiču za različite pomake.

SubBytes() Transformacija stanja koja primjenjuje S-tablica neovisno na svaki bajt stanja.

SubWord() Transformacija riječi u kojoj se S-tablica primjenjuje na svaki od četiri bajta riječi.

1.4 Parametri i simboli algoritma

b^{-1} Multiplikativni inverz elementa b u GF(2^8).

$\tilde{b}$ Ulaz u afinu transformaciju u AES S-kutiji.

dw Niz riječi za raspored ključeva koji je ulaz u ekvivalentnu inverznu šifru.

GF(2) Konačno polje s dva elementa.

GF(2^8) Konačno polje s 256 elemenata.

in Ulazni podaci u Cipher() ili InvCipher(), predstavljeni kao niz od 16 bajtova indeksiranih od 0 do 15.

$m(x)$ Modulus specificiran u ovom standardu za polinomski prikaz bajtova kao elemenata GF(2^8).

key Niz od Nk riječi koje čine ključ za AES-128, AES-192 ili AES-256.

N_b Broj stupaca koji čine stanje, gdje je svaki stupac 32-bitna riječ. Za ovaj standard, $N_b = 4$.

N_k Broj 32-bitnih riječi koje čine ključ. N_k je dodijeljen vrijednostima 4, 6 i 8 za AES-128, AES-192 i AES-256, redom (vidi Poglavlje 6.3).

N_r Broj rundi. N_r je dodijeljen redom vrijednostima 10, 12 i 14 za AES-128, AES-192 i AES-256.

out Izlazni podaci iz Cipher() ili InvCipher(), predstavljeni kao niz od 16 bajtova indeksiranih od 0 do 15.

R_{con} Niz riječi za konstante runde.

state Stanje, predstavljeno kao dvodimenzionalni niz od 16 bajtova, s redovima i stupcima indeksiranim od 0 do 3.

u[i] Za jednodimenzionalni niz u riječi ili bajtova, element u nizu koji je indeksiran nenegativnim cijelim brojem i.

u[i...i+3] Za niz u riječi, slijed u[i], u[i+1], u[i+2], u[i+3].

w Niz riječi za raspored ključeva.

$\oplus$ **(Isključivo ili)** operacija na bitovima, bajtovima ili na riječima, nadalje označeno kao XOR

$\cdot$ Množenje u GF(2^8).

$*$ Cjelobrojno množenje.

$\leftarrow$ Dodjela varijable u pseudokodu.

2 Oznake i konvencije

2.1 Ulazi i izlazi

Bit je binarna znamenka - 0 ili 1. Blok je niz od 128 bitova; ulazni i izlazni podaci za AES blok šifre su blokovi. Drugi ulaz za AES blok šifre, nazvan ključ, je niz bitova koji se obično unaprijed uspostavlja i održava kroz mnoge pozive blok šifre. Duljine ključeva za AES-128, AES-192 i AES-256 su redom 128, 192 i 256 bitova.

2.2 Bajtovi

Osnovna jedinica obrade u AES algoritmima je bajt — niz od osam bitova. Vrijednost bajta označava se konkatenacijom osam bitova između vitičastih zagrada, npr. {10100011}. Kada su bitovi bajta označeni indeksiranim varijablama, konvencija u ovom standardu je da indeksi opadaju slijeva nadesno, tj. $\{b_7\ b_6\ b_5\ b_4\ b_3\ b_2\ b_1\ b_0\}$.

Također je zgodno označiti vrijednosti bajtova koristeći heksadecimalnu notaciju. Šesnaest heksadecimalnih znakova predstavlja nizove od četiri bita, kao što je navedeno u Tablici 1. Bajt je predstavljen uređenim parom heksadecimalnih znakova, gdje lijevi znak u paru predstavlja četiri krajnja lijeva bita (tj. b_7, b_6, b_5, b_4), a desni znak u paru predstavlja četiri krajnja desna bita (tj. b_3, b_2, b_1, b_0). Na primjer, heksadecimalni oblik bajta {10100011} je {a3}.

Sequence	0000	0001	0010	0011	0100	0101	0110	0111
Character	0	1	2	3	4	5	6	7
Sequence	1000	1001	1010	1011	1100	1101	1110	1111
Character	8	9	a	b	c	d	e	f

Tablica 1: Heksadecimalni prikaz 4-bitnih nizova

2.3 Indeksiranje nizova bajtova

Kako bi se nedvosmisleno predstavili ulazni podaci i ključ kao nizovi bajtova, u ovom standardu usvojena je sljedeća konvencija indeksiranja. Za dani niz od 8k bitova,

$$r_0\, r_1\, r_2\, \ldots\, r_{(8k-3)}\, r_{(8k-2)}\, r_{(8k-1)}$$

(za neki pozitivni cijeli broj k), bajtovi a_j za $0 \leq j \leq k-1$ definirani su kao:

$$a_j = \{r_{8j}\, r_{(8j+1)}\, \ldots\, r_{(8j+7)}\}.$$

Dakle, na primjer, blok podataka

$$r_0\, r_1\, r_2\, \ldots\, r_{125}\, r_{126}\, r_{127}$$

je predstavljen nizom bajtova

$$a_0\, a_1\, a_2\, \ldots\, a_{13}\, a_{14}\, a_{15},$$

gdje

$$
\begin{aligned}
a_0 &= \{r_0\, r_1\, \ldots\, r_7\};\\
a_1 &= \{r_8\, r_9\, \ldots\, r_{15}\};\\
&\vdots\\
a_{15} &= \{r_{120}\, r_{121}\, \ldots\, r_{127}\}.
\end{aligned}
$$

Kao što je opisano u Poglavlju 2.2, bitovi unutar pojedinog bajta su indeksirani opadajućim redoslijedom slijeva nadesno. Ovaj redoslijed je prirodniji za aritmetiku konačnog polja na bajtovima koja je opisana u Poglavlju 3. Dvije vrste bitnih indeksa za nizove bajtova prikazane su u Slici 1.

Bit index in sequence	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	...
Byte index	0								1								...
Bit index in byte	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	...

Slika 1: Bajt i bit indeksi. Izvor: [2]

2.4 Stanje

Interno, algoritmi za AES blok šifre se izvode na dvodimenzionalnom (četiri po četiri) nizu bajtova zvanom stanje. U nizu stanja, označenom sa s, svaki pojedini bajt ima dva indeksa: indeks reda r u rasponu $0 \leq r < 4$ i indeks stupca c u rasponu $0 \leq c < 4$. Pojedini bajt stanja označava se s $s_{r,c}$ ili $s[r, c]$.

U pojedinostima za AES algoritme blok šifri u Poglavlju 4, prvi korak je kopiranje ulaznog niza bajtova $\text{in}_0, \text{in}_1, \ldots, \text{in}_{15}$ u niz stanja s na sljedeći način:

$$s[r, c] = \text{in}[r + 4c] \quad \text{za} \quad 0 \leq r < 4 \quad \text{i} \quad 0 \leq c < 4.$$

Zatim se na niz stanja primjenjuje niz transformacija, nakon čega se njegova konačna vrijednost kopira u izlazni niz bajtova $\text{out}_0, \text{out}_1, \ldots, \text{out}_{15}$ na sljedeći način:

$$\text{out}[r + 4c] = s[r, c] \quad \text{za} \quad 0 \leq r < 4 \quad \text{i} \quad 0 \leq c < 4.$$

Podudarnost između indeksa ulaza i izlaza s indeksima niza stanja prikazana je na Slici 2.

input bytes					*state array*				*output bytes*			
in_0	in_4	in_8	in_{12}	→	$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$	→	out_0	out_4	out_8	out_{12}
in_1	in_5	in_9	in_{13}		$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$		out_1	out_5	out_9	out_{13}
in_2	in_6	in_{10}	in_{14}		$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$		out_2	out_6	out_{10}	out_{14}
in_3	in_7	in_{11}	in_{15}		$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$		out_3	out_7	out_{11}	out_{15}

Slika 2: Ulaz i izlaz niza stanja. Izvor: [2]

2.5 Nizovi riječi

Riječ je niz od četiri bajta; blok se sastoji od četiri riječi. Četiri stupca niza stanja s interpretiraju se kao niz v od četiri riječi na sljedeći način, u notaciji iz Slike 2:

$$v_0 = \begin{pmatrix} s_{0,0} \\ s_{1,0} \\ s_{2,0} \\ s_{3,0} \end{pmatrix}, \quad v_1 = \begin{pmatrix} s_{0,1} \\ s_{1,1} \\ s_{2,1} \\ s_{3,1} \end{pmatrix}, \quad v_2 = \begin{pmatrix} s_{0,2} \\ s_{1,2} \\ s_{2,2} \\ s_{3,2} \end{pmatrix}, \quad v_3 = \begin{pmatrix} s_{0,3} \\ s_{1,3} \\ s_{2,3} \\ s_{3,3} \end{pmatrix}.$$

Dakle, indeks stupca c od s postaje indeks za v, a indeks reda r od s postaje indeks za četiri bajta u svakoj riječi.

Za dani jednodimenzionalni niz u riječi, $u[i]$ označava riječ koja je indeksirana s i, a niz od četiri riječi $u[i], u[i+1], u[i+2], u[i+3]$ označava se kao $u[i..i+3]$.

3 Matematički preduvjeti

Za neke transformacije AES algoritma opisane u Poglavlju 4, svaki bajt u nizu stanja interpretira se kao jedan od 256 elemenata konačnog polja, također poznatog kao Galoisovo polje, označenog kao GF(2^8).

Kako bi se definiralo zbrajanje i množenje u GF(2^8), svaki bajt {b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0} interpretira se kao polinom, označen s $b(x)$, na sljedeći način:

$$b(x) = b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0.$$

Na primjer, {01100011} je predstavljen polinomom $x^6 + x^5 + x + 1$.

3.1 Zbrajanje u GF(2^8)

Da bi se zbrojila dva elementa u konačnom polju GF(2^8), koeficijenti polinoma koji predstavljaju elemente zbrajaju se modulo 2 (tj. XOR operacijom, označenom sa $\oplus$), tako da $1 \oplus 1 = 0$, $1 \oplus 0 = 1$ i $0 \oplus 0 = 0$.

Ekvivalentno, dva bajta mogu se zbrojiti primjenom XOR operacijom na svaki par odgovarajućih bitova u bajtovima. Dakle, zbroj {a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0} i {b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0} je {$a_7 \oplus b_7$ $a_6 \oplus b_6$ $a_5 \oplus b_5$ $a_4 \oplus b_4$ $a_3 \oplus b_3$ $a_2 \oplus b_2$ $a_1 \oplus b_1$ $a_0 \oplus b_0$}. (U Poglavlju 4.1.4, ova definicija se proširuje na riječi.)

Na primjer, sljedeće tri reprezentacije zbrajanja su ekvivalentne:

$$(x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) = x^7 + x^6 + x^4 + x^2 \quad \text{(polinom)}$$

$$\{01010111\} \oplus \{10000011\} = \{11010100\} \quad \text{(binarno)}$$

$$\{57\} \oplus \{83\} = \{d4\} \quad \text{(heksadecimalno)}.$$

Zbog toga što se koeficijenti polinoma reduciraju modulo 2, koeficijent 1 je ekvivalentan koeficijentu -1, pa je zbrajanje ekvivalentno oduzimanju. Na primjer, $x^4 + x^2$ predstavlja isti element konačnog polja kao $x^4 - x^2$ i $-x^4 + x^2$. Slično, zbroj bilo kojeg elementa sa samim sobom je nulti element.

3.2 Množenje u GF(2^8)

Simbol $\cdot$ označava množenje u GF(2^8). Konceptualno, ovo množenje je definirano na dva bajta u dva koraka: 1) polinomi koji predstavljaju bajtove se množe kao polinomi, i 2) rezultirajući polinom se reducira modulo sljedeći fiksni polinom:

$$m(x) = x^8 + x^4 + x^3 + x + 1.$$

Unutar oba koraka, pojedini koeficijenti polinoma se reduciraju modulo 2.

Dakle, ako $b(x)$ i $c(x)$ predstavljaju bajtove b i c, tada je $b \cdot c$ predstavljen sljedećom modularnom redukcijom njihovog produkta kao polinoma:

$$b(x)c(x) \mod m(x).$$

Modularna redukcija pomoću $m(x)$ može se primijeniti na međukorake u računanju $b(x)c(x)$; stoga, korisno je razmotriti poseban slučaj kada je $c(x) = x$ (tj. $c = \{02\}$). Konkretno, produkt $b \cdot \{02\}$ može se izraziti kao funkcija b, označena s XTIMES(b), na sljedeći način:

$$\text{XTIMES}(b) = \begin{cases} \{b_6\, b_5\, b_4\, b_3\, b_2\, b_1\, b_0\, 0\} & \text{ako } b_7 = 0 \\ \{b_6\, b_5\, b_4\, b_3\, b_2\, b_1\, b_0\, 0\} \oplus \{0\,0\,0\,1\,1\,0\,1\,1\} & \text{ako } b_7 = 1 \end{cases}.$$

Množenje višim potencijama od x (kao što su $\{04\}$, $\{08\}$ i $\{10\}$) može se implementirati ponovljenom primjenom XTIMES(). Na primjer, neka je $b = \{57\}$:

$$\begin{aligned} \{57\} \cdot \{01\} &= \{57\} \\ \{57\} \cdot \{02\} &= \text{XTIMES}(\{57\}) = \{ae\} \\ \{57\} \cdot \{04\} &= \text{XTIMES}(\{ae\}) = \{47\} \\ \{57\} \cdot \{08\} &= \text{XTIMES}(\{47\}) = \{8e\} \\ \{57\} \cdot \{10\} &= \text{XTIMES}(\{8e\}) = \{07\} \\ \{57\} \cdot \{20\} &= \text{XTIMES}(\{07\}) = \{0e\} \\ \{57\} \cdot \{40\} &= \text{XTIMES}(\{0e\}) = \{1c\} \\ \{57\} \cdot \{80\} &= \text{XTIMES}(\{1c\}) = \{38\}. \end{aligned}$$

Ovaj rezultat olakšava računanje bilo kojeg umnoška od $\{57\}$. Na primjer, budući da je $\{13\} = \{10\} \oplus \{02\} \oplus \{01\}$, slijedi da je

$$\begin{aligned} \{57\} \cdot \{13\} &= \{57\} \cdot (\{01\} \oplus \{02\} \oplus \{10\}) \\ &= \{57\} \oplus \{ae\} \oplus \{07\} \\ &= \{fe\}. \end{aligned}$$

3.3 Množenje riječi s fiksnom matricom

Dvije transformacije - *MixColumns()* i *InvMixColumns()* – u algoritmima za AES blok šifre mogu se izraziti u terminima matričnog množenja. Konkretno, za svaku transformaciju specificirana je posebna fiksna matrica. Za obje matrice, svaki od 16 unosa matrice je bajt jedne specificirane riječi, označene ovdje kao $[a_0, a_1, a_2, a_3]$.

Za danu ulaznu riječ $[b_0, b_1, b_2, b_3]$ za transformaciju, izlazna riječ $[d_0, d_1, d_2, d_3]$ određena je aritmetikom konačnog polja na sljedeći način:

$$\begin{aligned} d_0 &= (a_0 \cdot b_0) \oplus (a_3 \cdot b_1) \oplus (a_2 \cdot b_2) \oplus (a_1 \cdot b_3) \\ d_1 &= (a_1 \cdot b_0) \oplus (a_0 \cdot b_1) \oplus (a_3 \cdot b_2) \oplus (a_2 \cdot b_3) \\ d_2 &= (a_2 \cdot b_0) \oplus (a_1 \cdot b_1) \oplus (a_0 \cdot b_2) \oplus (a_3 \cdot b_3) \\ d_3 &= (a_3 \cdot b_0) \oplus (a_2 \cdot b_1) \oplus (a_1 \cdot b_2) \oplus (a_0 \cdot b_3). \end{aligned}$$

Matrični oblik prijašnje jednadžbe je:

$$\begin{pmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{pmatrix} = \begin{pmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix}.$$

3.4 Multiplikativni inverzi u GF(2^8)

Za bajt $b \neq \{00\}$, njegov multiplikativni inverz je jedinstveni bajt, označen kao b^{-1}, tako da je

$$b \cdot b^{-1} = \{01\}.$$

Definicija transformacije *SubBytes()* u specifikacijama AES blok šifre uključuje multiplikativne inverze u GF(2^8), koji se mogu izračunati na sljedeći način:

$$b^{-1} = b^{254}.$$

Alternativno, neka je $b(x)$ polinom koji predstavlja b. Prošireni Euklidov algoritam može se primijeniti na $b(x)$ i $m(x)$ kako bi se pronašli polinomi $a(x)$ i $c(x)$ tako da je

$$b(x)a(x) + m(x)c(x) = 1.$$

Iz toga slijedi da je $a(x)$ polinom koji predstavlja b^{-1}.

4 Specifikacija algoritma

Opća funkcija za izvršavanje AES-128, AES-192 ili AES-256 označena je kao *Cipher()*; njen inverz je označen kao *InvCipher()*.

Suština algoritama za *Cipher()* i *InvCipher()* je niz fiksnih transformacija stanja nazvanih runda. Svaka runda zahtijeva dodatni ulaz nazvan ključ runde; ključ runde je blok koji se obično predstavlja kao niz od četiri riječi (tj. 16 bajtova).

Rutina za proširenje ključa, označena kao *KeyExpansion()*, uzima ključ blok šifre kao ulaz i generira ključeve runde kao izlaz. Konkretno, ulaz za *KeyExpansion()* predstavljen je kao niz riječi, označen kao *key*, a izlaz je prošireni niz riječi, označen kao w, nazvan raspored ključeva.

Blok šifre AES-128, AES-192 i AES-256 razlikuju se u tri aspekta: 1) duljina ključa; 2) broj rundi, koji određuje veličinu potrebnog rasporeda ključeva; i 3) specifikacija rekurzije unutar *KeyExpansion()*. Za svaki algoritam, broj rundi označen je kao N_r, a broj riječi ključa označen je kao N_k. (Broj riječi u stanju označen je kao N_b za Rijndael općenito; u ovom standardu, $N_b = 4$.) Specifične vrijednosti N_k, N_b i N_r dane su u Slici 3. Nijedna druga konfiguracija Rijndaela ne odgovara ovom standardu.

U poglavlju 5.3 opisani su implementacijski problemi vezani uz duljinu ključa, veličinu bloka i broj rundi.

Tri ulaza za *Cipher()* su: 1) ulazni podaci *in* su blok predstavljen kao linearni niz od 16 bajtova; 2) broj rundi N_r za instancu; i 3) ključ te runde. Dakle,

	Key length		Block size		Number of rounds
	Nk	(in bits)	*Nb*	(in bits)	*Nr*
AES-128	4	128	4	128	10
AES-192	6	192	4	128	12
AES-256	8	256	4	128	14

Slika 3: Kombinacije ključ-blok-runda. Izvor: [2]

$$\begin{aligned} \text{AES-128}(\text{in}, \text{key}) &= \text{Cipher}(\text{in}, 10, \text{KeyExpansion}(\text{key})) \\ \text{AES-192}(\text{in}, \text{key}) &= \text{Cipher}(\text{in}, 12, \text{KeyExpansion}(\text{key})) \\ \text{AES-256}(\text{in}, \text{key}) &= \text{Cipher}(\text{in}, 14, \text{KeyExpansion}(\text{key})). \end{aligned} \quad (5.1)$$

Inverzne permutacije definirane su zamjenom *Cipher()* s *InvCipher()* u poglavlju 4.1.

Specifikacije *Cipher()*, *KeyExpansion()* i *InvCipher()* redom su dane u Poglavljima 4.1, 4.6 i 4.7.

4.1 Cipher()

Runde u specifikaciji *Cipher()* sastoje se od sljedeće četiri transformacije usmjerene na bajtove unutar stanja:

- *SubBytes()* primjenjuje zamjensku tablicu (S-tablicu) na svaki bajt.
- *ShiftRows()* pomiče redove niza stanja za različite pomake.
- *MixColumns()* miješa podatke unutar svakog stupca stanja.

- *AddRoundKey()* kombinira ključeve runde za pojedino stanja.

Četiri transformacije su specificirane u Poglavljima 4.1.1–4.1.4. U tim specifikacijama, transformirani bit, bajt ili blok označen je dodavanjem simbola ' kao superskripta na originalnu varijablu, tj. b_i', b', $s_{i,j}'$ ili s'.

Ključevi runde za *AddRoundKey()* generiraju se pomoću *KeyExpansion()*, koja je specificirana u Poglavlju 4.2. Konkretno, raspored ključeva predstavljen je kao niz w od $4 \times (N_r + 1)$ riječi.

Algoritam 1: Pseudokod za CIPHER()

ulaz : in, N_r, w
izlaz: state

```
1  state ← in;
2  state ← AddRoundKey(state, w[0..3]);
3  for round ← 1 to N_r − 1 do
4  |   state ← SubBytes(state);
5  |   state ← ShiftRows(state);
6  |   state ← MixColumns(state);
7  |   state ← AddRoundKey(state, w[4 * round .. 4 * round + 3]);
8  state ← SubBytes(state);
9  state ← ShiftRows(state);
10 state ← AddRoundKey(state, w[4 * N_r .. 4 * N_r + 3]);
11 return state;
```

Prvi korak (Linija 2) je kopiranje ulaza u niz stanja koristeći konvencije iz Poglavlja 2.4. Nakon inicijalnog dodavanja ključa runde (Linija 3), niz stanja se transformira s N_r primjena funkcije runde (Linije 4–12); konačna runda (Linije 10–12) se razlikuje po tome što je transformacija *MixColumns()* izostavljena. Konačno stanje se zatim vraća kao izlaz (Linija 13), kako je opisano u Poglavlju 2.4.

4.1.1 SubBytes()

SubBytes() je inverzibilna, nelinearna transformacija stanja u kojoj se zamjenska tablica, nazvana S-tablica, primjenjuje neovisno na svaki bajt u stanju. AES S-tablica je označena kao *SBOX()*.

Neka je b ulazni bajt za *SBOX()*, a neka je c konstantni bajt $\{01100011\}$. Izlazni bajt $b' = SBOX(b)$ se konstruira kompozicijom sljedeće dvije transformacije:

1. Definirajte međuvrijednost $\tilde{b}$, na sljedeći način, gdje je b^{-1} multiplikativni inverz od b, kako je opisano u Poglavlju 2.4:

$$\tilde{b} = \begin{cases} \{00\} & \text{ako } b = \{00\} \\ b^{-1} & \text{ako } b \neq \{00\} \end{cases}$$

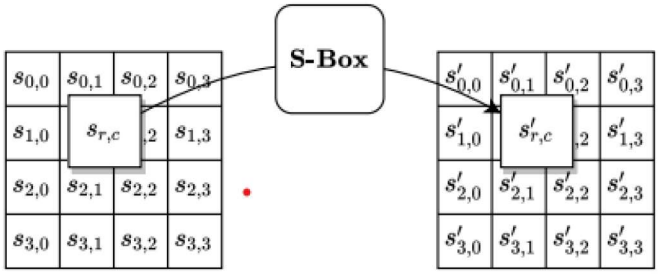
2. Primjenite sljedeću affin transformaciju na bitove $\tilde{b}$ kako biste proizveli bitove b':

$$b_i' = \tilde{b}_i \oplus \tilde{b}_{(i+4) \mod 8} \oplus \tilde{b}_{(i+5) \mod 8} \oplus \tilde{b}_{(i+6) \mod 8} \oplus \tilde{b}_{(i+7) \mod 8} \oplus c_i.$$

Matrica prijašnje jednadžbe dana je ispod:

$$\begin{pmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} \tilde{b}_0 \\ \tilde{b}_1 \\ \tilde{b}_2 \\ \tilde{b}_3 \\ \tilde{b}_4 \\ \tilde{b}_5 \\ \tilde{b}_6 \\ \tilde{b}_7 \end{pmatrix} \oplus \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}.$$

Slika 4 prikazuje kako SubBytes transformira stanje.



Slika 4: Prikaz SubBytes. Izvor: [2]

AES S-tablica predstavljena je u heksadecimalnom obliku u tablici 2. Na primjer, ako je $s_{r,c} = \{53\}$, tada bi vrijednost zamjene bila određena presjekom retka s indeksom '5' i stupca s indeksom '3' u tablici 2, tako da je $s'_{r,c} = \{ed\}$.

	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

Tablica 2: *SBOX()*: vrijednosti zamjene za bajt xy (u heksadecimalnom formatu)

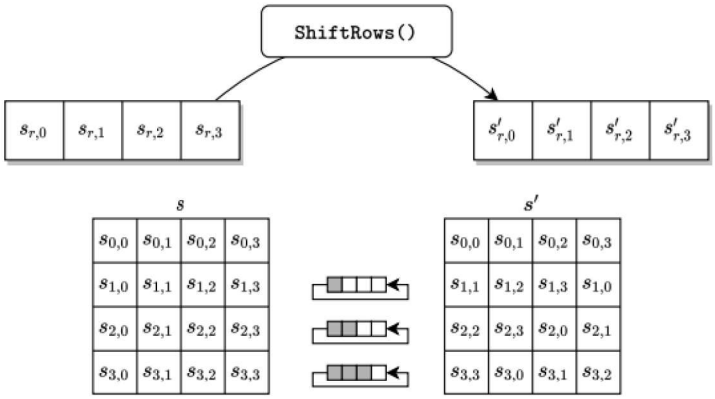
4.1.2 ShiftRows()

ShiftRows() je transformacija stanja u kojoj se bajtovi u posljednja tri reda stanja ciklički pomiču. Broj pozicija za koje se bajtovi pomiču ovisi o indeksu reda r, na sljedeći način:

$$s'_{r,c} = s_{r,(c+r) \mod 4} \quad \text{za} \quad 0 \leq r < 4 \quad \text{i} \quad 0 \leq c < 4.$$

ShiftRows() je ilustriran na Slici 5. U toj reprezentaciji stanja, efekt je pomicanje svakog bajta za r pozicija ulijevo u redu, ciklički pomičući krajnjih r bajtova na desni kraj reda. Prvi red, gdje je $r = 0$, ostaje nepromijenjen.

Slika 5 prikazuje kako ShiftRows transformira stanje



Slika 5: Ilustracija ShiftRows. Izvor: [2]

4.1.3 MixColumns()

MixColumns() je transformacija stanja koja množi svaki od četiri stupca stanja s jednom fiksnom matricom, kako je opisano u Poglavlju 3.3, s unosima uzetim iz sljedeće riječi:

$$[a_0, a_1, a_2, a_3] = [\{02\}, \{01\}, \{01\}, \{03\}].$$

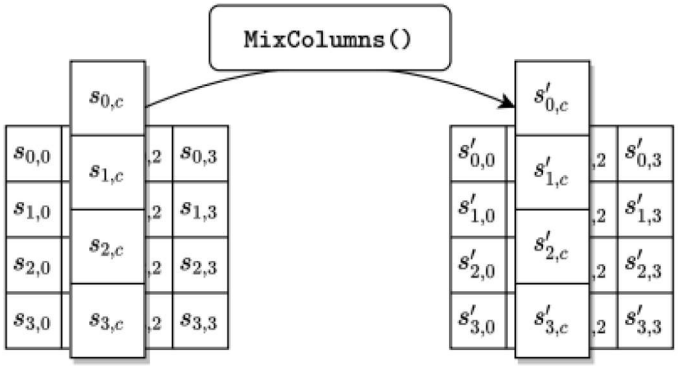
Dakle,

$$\begin{pmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{pmatrix} \quad \text{za} \quad 0 \leq c < 4,$$

tako da su pojedini izlazni bajtovi definirani na sljedeći način:

$$\begin{aligned} s'_{0,c} &= (\{02\} \cdot s_{0,c}) \oplus (\{03\} \cdot s_{1,c}) \oplus s_{2,c} \oplus s_{3,c} \\ s'_{1,c} &= s_{0,c} \oplus (\{02\} \cdot s_{1,c}) \oplus (\{03\} \cdot s_{2,c}) \oplus s_{3,c} \\ s'_{2,c} &= s_{0,c} \oplus s_{1,c} \oplus (\{02\} \cdot s_{2,c}) \oplus (\{03\} \cdot s_{3,c}) \\ s'_{3,c} &= (\{03\} \cdot s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (\{02\} \cdot s_{3,c}). \end{aligned}$$

Slika 6 prikazuje MixColumns transformira stanje



Slika 6: Prikaz MixColumns. Izvor: [2]

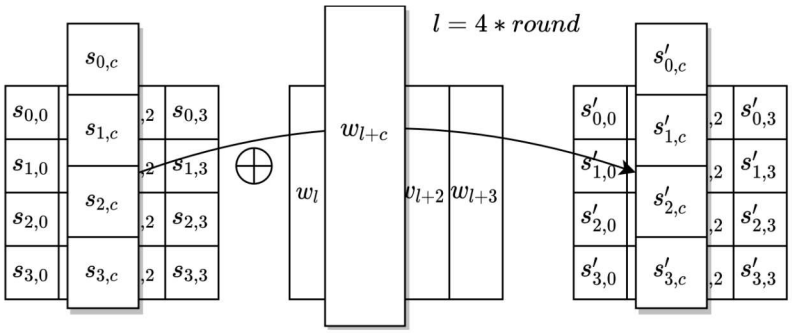
4.1.4 AddRoundKey()

AddRoundKey() je transformacija stanja u kojoj se ključ runde kombinira sa stanjem primjenom bitovne XOR operacije. Konkretno, svaki ključ runde se sastoji od četiri riječi iz rasporeda ključeva (opisanog u Poglavlju 4.2), od kojih se svaka kombinira sa stupcem stanja na sljedeći način:

$$[s'_{0,c}, s'_{1,c}, s'_{2,c}, s'_{3,c}] = [s_{0,c}, s_{1,c}, s_{2,c}, s_{3,c}] \oplus [w_{4\cdot\text{round}+c}] \quad \text{za} \quad 0 \leq c < 4 \quad (5.9)$$

gdje je *round* vrijednost u rasponu $0 \leq round \leq N_r$, a $w[i]$ je niz riječi iz rasporeda ključeva opisanog u Poglavlju 4.2. U specifikaciji *Cipher()*, *AddRoundKey()* se poziva $N_r + 1$ puta - jednom prije prve primjene funkcije runde (vidi Algoritam 1) i jednom unutar svake od N_r rundi, kada $1 \leq round \leq N_r$.

Djelovanje ove transformacije ilustrirano je na Slici 7, gdje je $l = 4 \cdot round$. Adresiranje bajtova unutar riječi rasporeda ključeva opisano je u Poglavlju 2.5.



Slika 7: Prikaz AddRoundKey. Izvor: [2]

4.2 KeyExpansion()

KeyExpansion() je rutina koja se primjenjuje na ključ za generiranje $4 \times (N_r + 1)$ riječi. Dakle, četiri riječi se generiraju za svaku od $N_r + 1$ primjena *AddRoundKey()* unutar specifikacije *Cipher()*, kako je opisano u Poglavlju 4.1.4. Izlaz rutine sastoji se od linearnog niza riječi, označenog sa $w[i]$, gdje je i u rasponu $0 \leq i < 4 \times (N_r + 1)$.

KeyExpansion() koristi 10 fiksnih riječi označenih kao *Rcon[j]* za $1 \leq j \leq 10$. Ovih 10 riječi nazivaju se konstante runde. Za AES-128, posebna rundska konstanta se koristi

pri generiranju svakog od 10 ključeva runde. Za AES-192 i AES-256, rutina za proširenje ključa koristi prvih osam odnosno sedam od ovih konstanti. Vrijednosti *Rcon[j]* dane su u heksadecimalnoj notaciji u Tablici 3:

j	*Rcon[j]*	j	*Rcon[j]*
1	[01, 00, 00, 00]	6	[20, 00, 00, 00]
2	[02, 00, 00, 00]	7	[40, 00, 00, 00]
3	[04, 00, 00, 00]	8	[80, 00, 00, 00]
4	[08, 00, 00, 00]	9	[1b, 00, 00, 00]
5	[10, 00, 00, 00]	10	[36, 00, 00, 00]

Tablica 3: Konstante runde

Vrijednost krajnjeg lijevog bajta *Rcon[j]* u polinomskom obliku je x^j. Za $j > 0$, ovi bajtovi mogu biti generirani sukcesivnom primjenom *XTIMES()* na bajt predstavljen sa x^{j-1}.

Dvije transformacije na riječima se koriste unutar *KeyExpansion()*: *RotWord()* i *SubWord()*. Za danu ulaznu riječ predstavljenu kao niz $[a_0, a_1, a_2, a_3]$ od četiri bajta,

$$RotWord([a_0, a_1, a_2, a_3]) = [a_1, a_2, a_3, a_0], \quad (5.10)$$

i

$$SubWord([a_0, \ldots, a_3]) = [SBOX(a_0), SBOX(a_1), SBOX(a_2), SBOX(a_3)]. \quad (5.11)$$

Proširenje ključa odvija se prema pseudokodu u Algoritmu 2. Prvih N_k riječi proširenog ključa su sam ključ. Svaka sljedeća riječ $w[i]$ generira se rekurzivno iz prethodne riječi, $w[i-1]$, i riječi N_k pozicija ranije, $w[i-N_k]$, na sljedeći način:

- Ako je i višekratnik N_k, tada $w[i] = w[i-N_k] \oplus SubWord(RotWord(w[i-1])) \oplus Rcon[i/N_k]$.
- Za AES-256, ako je $i+4$ višekratnik 8, tada $w[i] = w[i-N_k] \oplus SubWord(w[i-1])$.
- Za sve ostale slučajeve, $w[i] = w[i-N_k] \oplus w[i-1]$.

Algoritam 2: Pseudokod za KeyExpansion()

```
   ulaz : ključ
   izlaz: w
1  i ← 0;
2  while i ≤ N_k − 1 do
3  |  w[i] ← ključ[4 ∗ i..4 ∗ i + 3];
4  |  i ← i + 1;
5  while i ≤ 4 ∗ N_r + 3 do
6  |  temp ← w[i − 1];
7  |  if i mod N_k == 0 then
8  |  |  temp ← SubWord(RotWord(temp)) ⊕ Rcon[i/N_k];
9  |  else if N_k > 6 i i mod N_k == 4 then
10 |  |  temp ← SubWord(temp);
11 |  w[i] ← w[i − N_k] ⊕ temp;
12 |  i ← i + 1;
13 return w;
```

4.3 InvCipher()

Za implcmcntaciju *InvCipher()*, transformacijc u spccifikaciji *Cipher()* (Poglavljc 4.1) su inverzne i izvršavaju se obrnutim redoslijedom.

Algoritam 3: Pseudokod za InvCipher()

```
   ulaz : in, N_r, w
   izlaz: state
1  state ← in;
2  state ← AddRoundKey(state, w[4 * N_r .. 4 * N_r + 3]);
3  for round ← N_r − 1 to 1 do
4  |  state ← InvShiftRows(state);
5  |  state ← InvSubBytes(state);
6  |  state ← AddRoundKey(state, w[4 * round .. 4 * round + 3]);
7  |  state ← InvMixColumns(state);
8  state ← InvShiftRows(state);
9  state ← InvSubBytes(state);
10 state ← AddRoundKey(state, w[0..3]);
11 return state;
```

Inverzne transformacije stanja — označene kao *InvShiftRows()*, *InvSubBytes()*, *InvMixColumns()* i *AddRoundKey()* — opisane su u Poglavljima 4.3.1–4.3.4.

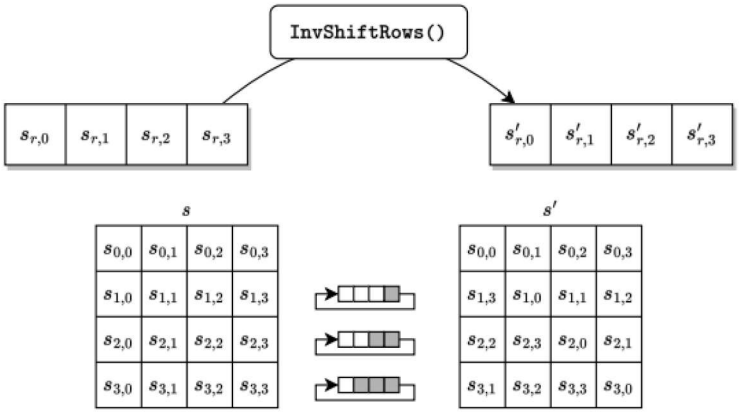
4.3.1 InvShiftRows()

InvShiftRows() je inverzija *ShiftRows()*. Konkretno, bajtovi u posljednja tri reda stanja ciklički se pomiču na sljedeći način:

$$s'_{r,c} = s_{r,(c-r) \mod 4} \quad \text{za} \quad 0 \leq r < 4 \quad \text{i} \quad 0 \leq c < 4. \quad (5.12)$$

InvShiftRows() je ilustriran na Slici 8. U toj reprezentaciji stanja, efekt je pomicanje

svakog bajta za r poziciju udesno u redu, ciklički pomičući krajnjih r bajtova na lijevi kraj reda. Prvi red, gdje je $r = 0$, ostaje nepromijenjen.



Slika 8: Ilustracija InvShiftRows. Izvor: [2]

4.3.2 InvSubBytes()

InvSubBytes() je inverzija *SubBytes()*, u kojoj se inverzija *SBOX()*, označena kao *InvSBOX()*, primjenjuje na svaki bajt stanja. *InvSBOX()* je izveden iz Tablice 2 zamjenom uloga ulaza i izlaza, kako je prikazano u Tablici 4:

	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

Tablica 4: *InvSBOX()*: Vrijednosti zamjene za bajt xy (u heksadecimalnom formatu)

4.3.3 InvMixColumns()

InvMixColumns() je inverzija *MixColumns()*. Konkretno, *InvMixColumns()* množi svaki od četiri stupca stanja s jednom fiksnom matricom, kako je opisano u Poglavlju 3.3, s unosima uzetim iz sljedeće riječi:

$$[a_0, a_1, a_2, a_3] = [\{0e\}, \{09\}, \{0d\}, \{0b\}]. \quad (5.13)$$

Dakle,

$$\begin{pmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{pmatrix} = \begin{pmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{pmatrix} \begin{pmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{pmatrix} \quad \text{za} \quad 0 \leq c < 4, \quad (5.14)$$

Kao rezultat ovog matričnog množenja, četiri bajta u stupcu zamijenjena su sljedećim:

$$\begin{aligned} s'_{0,c} &= (\{0e\} \cdot s_{0,c}) \oplus (\{0b\} \cdot s_{1,c}) \oplus (\{0d\} \cdot s_{2,c}) \oplus (\{09\} \cdot s_{3,c}) \\ s'_{1,c} &= (\{09\} \cdot s_{0,c}) \oplus (\{0e\} \cdot s_{1,c}) \oplus (\{0b\} \cdot s_{2,c}) \oplus (\{0d\} \cdot s_{3,c}) \\ s'_{2,c} &= (\{0d\} \cdot s_{0,c}) \oplus (\{09\} \cdot s_{1,c}) \oplus (\{0e\} \cdot s_{2,c}) \oplus (\{0b\} \cdot s_{3,c}) \\ s'_{3,c} &= (\{0b\} \cdot s_{0,c}) \oplus (\{0d\} \cdot s_{1,c}) \oplus (\{09\} \cdot s_{2,c}) \oplus (\{0e\} \cdot s_{3,c}). \end{aligned}$$

4.3.4 Inverzija AddRoundKey()

AddRoundKey(), opisana u Poglavlju 4.1.4, je svoj vlastiti inverz.

5 Razmatranja pri Implementaciji

U ovom poglavlju bit će razmatrani ključni implementacijski aspekti AES algoritma. Razmotrit će se zahtjevi za duljinu ključa, ograničenja u ključevima, proširenja parametara te preporuke za implementaciju algoritma na različitim platformama. Cilj je pružiti uvid u praktične izazove i smjernice za implementaciju AES-a, uzimajući u obzir performanse, sigurnosne aspekte i mogućnosti interoperabilnosti.

5.1 Key Length Requirements

Implementacija AES algoritma mora podržavati barem jednu od tri duljine ključa specificirane u Poglavlju 4: 128, 192 ili 256 bita (tj. $N_k = 4, 6$ ili 8, redom). Implementacije mogu opcionalno podržavati dvije ili tri duljine ključa, što može promicati interoperabilnost implementacije algoritma.

5.2 Keying Restrictions

Kada je kriptografski ključ generiran na odgovarajući način, nema ograničenja pri korištenju rezultirajućeg ključa za AES algoritam.

5.3 Parameter Extensions

Na Slici 3, ovaj Standard eksplicitno definira dopuštene vrijednosti za duljinu ključa (N_k), veličinu bloka (N_b) i broj rundi (N_r). Međutim, buduće revizije ovog Standarda mogle bi uključivati promjene ili dodatke dopuštenih vrijednosti za te parametre.

5.4 Implementation Suggestions Regarding Various Platforms

Moguće su varijacije u implementaciji koje u mnogim slučajevima mogu ponuditi prednosti u performansama ili drugim aspektima. Za zadani isti ulazni ključ i podatke (čisti tekst ili šifrat), bilo koja implementacija koja proizvodi isti izlaz (šifrat ili čisti tekst) kao algoritam specificiran u ovom Standardu je ekvivalentna implementacija AES algoritma.

Dokument prijedloga AES-a i drugi resursi smješteni na AES stranici uključuju prijedloge o tome kako učinkovito implementirati AES algoritam na raznim platformama. Predložene implementacije namijenjene su objašnjavanju unutarnjeg funkcioniranja AES algoritma, ali ne pružaju zaštitu od raznih napada na implementaciju.

Fizička implementacija može otkriti informacije o ključu putem bočnih kanala, kao što je vrijeme potrebno za izvršenje izračuna, ili kada se u izračun ubrizgavaju greške. Kada su takvi napadi neinvazivni, mogu biti učinkoviti čak i kada postoje mehanizmi za otkrivanje fizičkog neovlaštenog pristupa uređaju. Na primjer, napadi temeljeni na vremenu pristupa predmemoriji mogu utjecati na AES implementacije na softverskim platformama koje koriste predmemoriju za ubrzavanje pristupa podacima iz glavne memorije.

Trebalo bi razmotriti zaštitu implementacija AES algoritma protiv napada na implementaciju gdje je to primjenjivo. Takva razmatranja su izvan opsega ovog rada, ali se uzimaju u obzir prilikom testiranja sukladnosti s algoritmom u ovom Standardu prema validacijskom programu razvijenom od strane NIST-a.

6 Modovi operacija

Modovi rada blok šifre su kriptografske funkcije koje koriste blok šifru za pružanje informatičkih usluga, kao što su povjerljivost i autentifikacija. NIST-preporučeni modovi rada specificirani su u seriji 800-38 NIST-ovih posebnih publikacija.

AES algoritam može se koristiti u različitim modovima operacije kako bi se zadovoljile specifične sigurnosne potrebe. Najpoznatiji modovi operacija su ECB, CBC, CFB, OFB i CTR. Svaki od ovih modova ima jedinstvene karakteristike, prednosti i nedostatke.

6.1 Padding za AES Algoritam

AES algoritam radi na blokovima podataka fiksne veličine, 16 bajtova (128 bitova). Kada se šifriraju podaci čija je duljina manja od 16 bajtova ili nije višekratnik 16, potrebno je primijeniti postupak podstavljanja (padding) kako bi se popunio posljednji blok. PKCS#7 je najčešće korišten standard za podstavljanje blokova u simetričnoj kriptografiji.

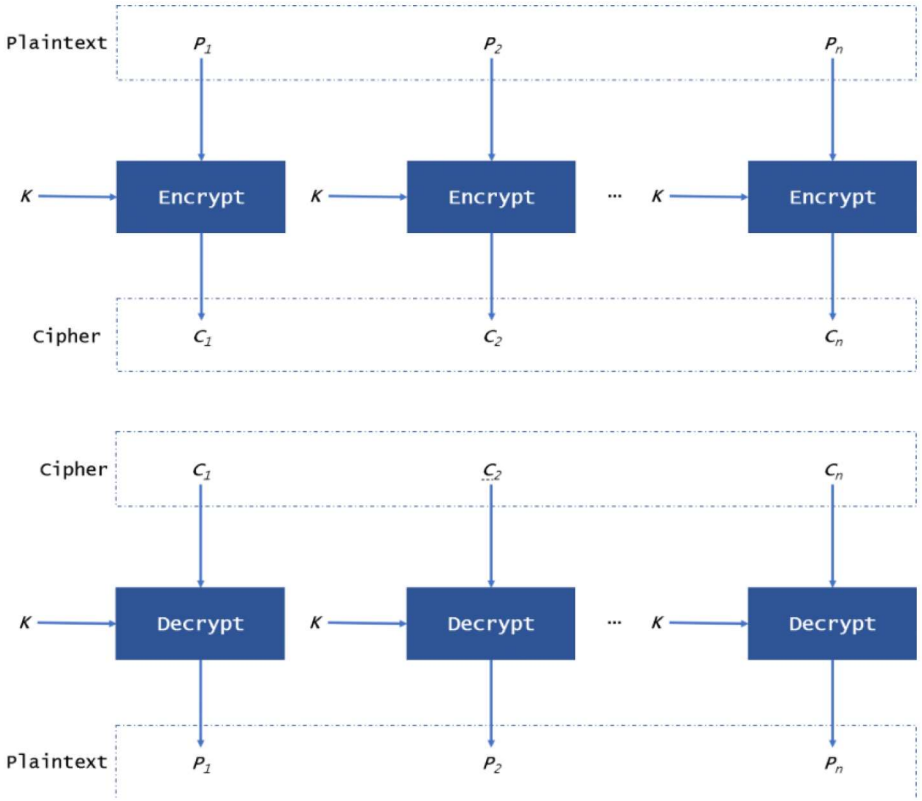
PKCS#7 padding dodaje niz bajtova na kraj podataka koji se šifriraju, gdje je svaki bajt jednak broju bajtova koji se dodaju. Na primjer, ako je potrebno dodati 4 bajta podstave, svaki od tih bajtova će imati vrijednost 0x04. Ako je duljina podataka već višekratnik od 16, cijeli blok podstave se dodaje s vrijednošću 0x10 (16 decimalno).

Ovaj pristup omogućava jednostavno uklanjanje podstave nakon dešifriranja. Proces dešifriranja uključuje provjeru vrijednosti zadnjeg bajta, koji određuje koliko bajtova podstave treba ukloniti. Ako podaci nisu pravilno podstavljeni, to ukazuje na grešku. PKCS#7 podstavljanje se koristi u modovima operacija poput CBC (Cipher Block Chaining) i ECB (Electronic Codebook), gdje je potrebno poravnanje podataka na veličinu bloka.

6.2 Electronic Codebook (ECB)

Electronic Codebook (ECB) je najjednostavniji mod operacije. U ECB modu, čisti tekst se dijeli na blokove, a svaki blok se šifrira zasebno koristeći isti ključ.

- **Prednosti:** Jednostavan za implementaciju i omogućuje paralelno šifriranje i dešifriranje blokova.
- **Nedostaci:** Nema difuzije, što znači da se isti blok čistog teksta uvijek šifrira u isti blok šifriranog teksta. Ovo može dovesti do obrazaca u šifriranom tekstu, što olakšava napade na šifrirane podatke.

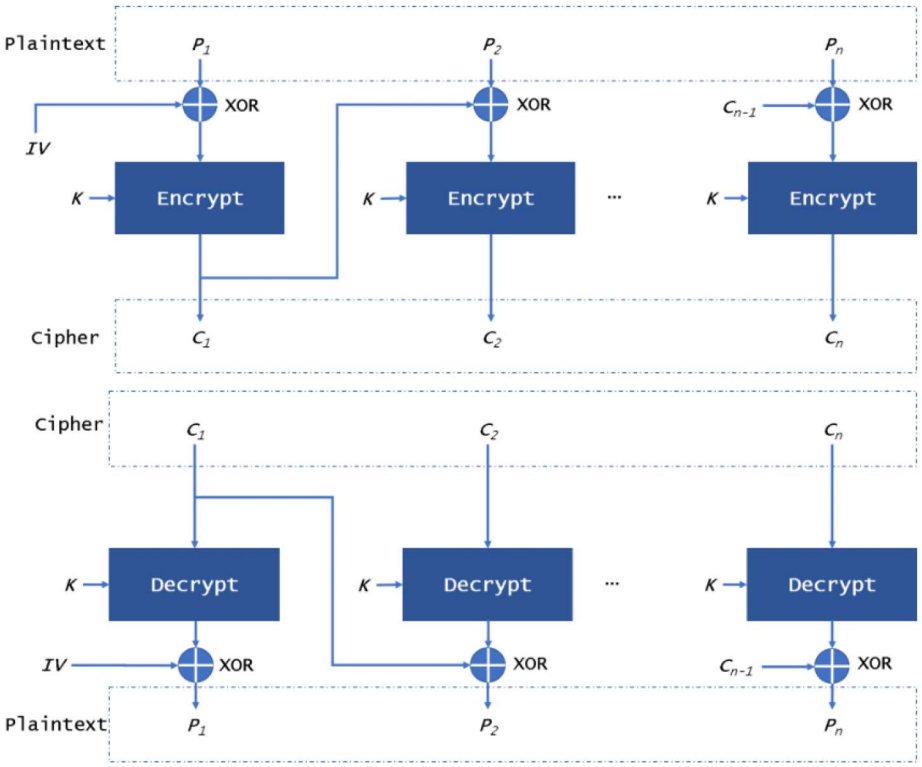


Slika 9: Ilustracija ECB. Izvor: [3]

6.3 Cipher Block Chaining (CBC)

Cipher Block Chaining (CBC) mod poboljšava sigurnost u odnosu na ECB tako što svaki blok čistog teksta koristi prethodni šifrirani blok za šifriranje sljedećeg bloka. Ovo dodaje sloj povezanosti između blokova i koristi početni vektor (IV) za inicijalizaciju.

- **Prednosti:** Pruža bolju sigurnost od ECB-a jer svaki blok ovisi o prethodnom bloku, čime se uklanjaju obrasci u šifriranom tekstu.
- **Nedostaci:** Šifriranje i dešifriranje moraju se izvoditi sekvencijalno, što je sporije od paralelnog pristupa.

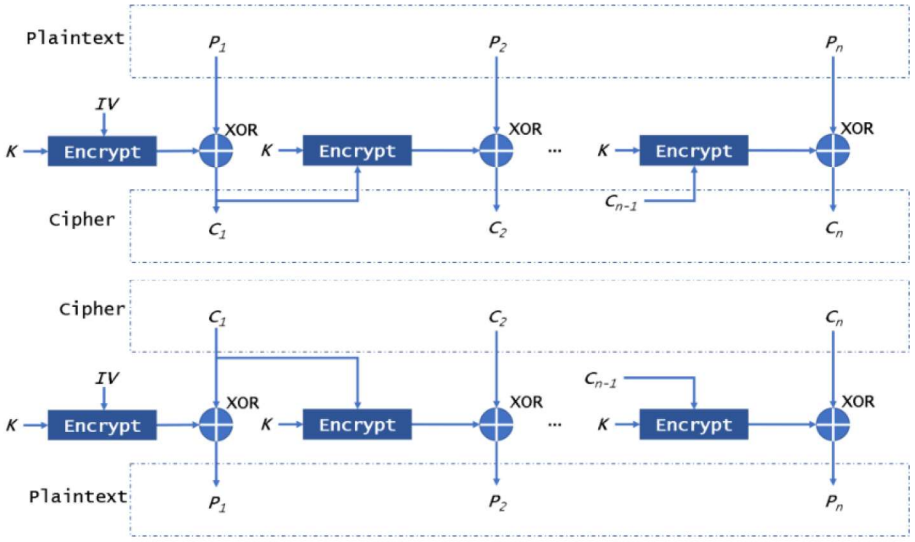


Slika 10: Ilustracija CBC. Izvor: [3]

6.4 Cipher Feedback (CFB)

Cipher Feedback (CFB) mod koristi šifrirane blokove za šifriranje sljedećih blokova čistog teksta, slično načinu na koji CBC mod koristi prethodni šifrirani blok. Međutim, dok CBC zahtijeva da se svaki blok šifrira sekvencijalno, CFB omogućava rad s podacima koji nisu poravnati na blokove i nudi samosinkronizaciju. U usporedbi s CBC-om, CFB može pružiti veću fleksibilnost u radu s dinamičnim podacima i omogućuje kontinuirano šifriranje bez potrebe za sinkronizacijom između pošiljatelja i primatelja.

- **Prednosti:** Pruža samosinkronizaciju; prijemnik može dešifrirati bez potrebe za sinkronizacijom pošiljatelja.
- **Nedostaci:** Sporije je od nekih drugih modova jer zahtijeva sekvencijalno šifriranje.

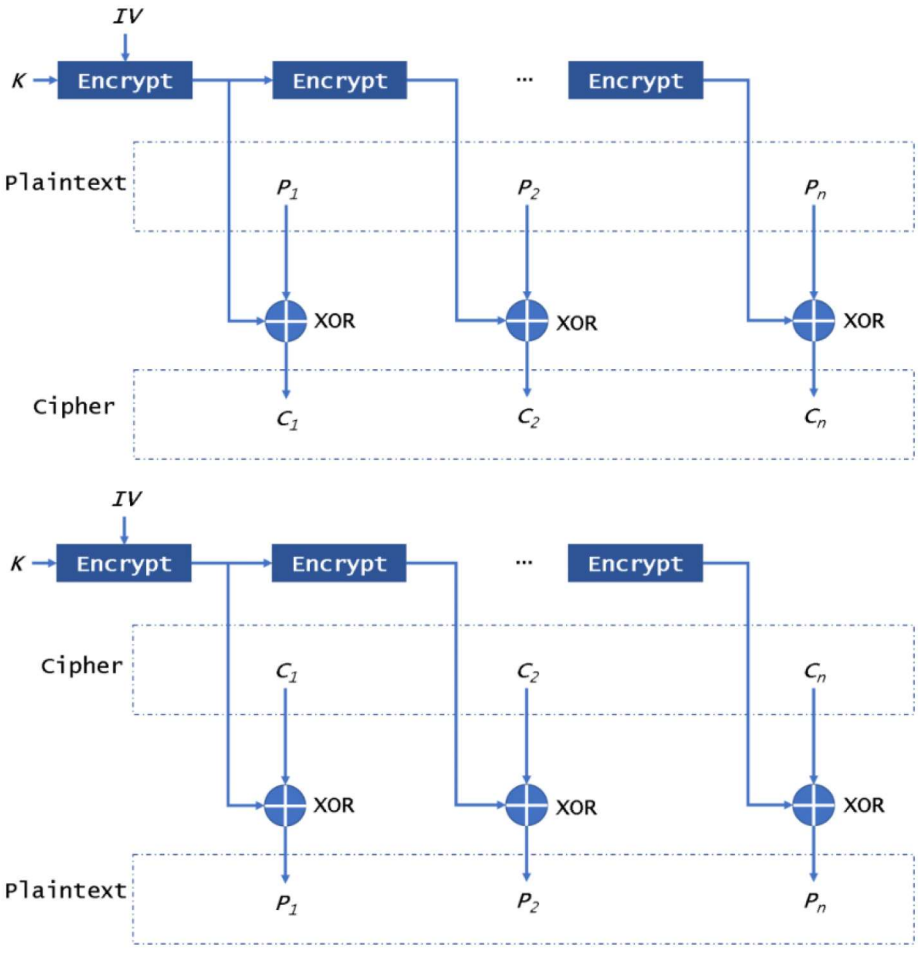


Slika 11: Ilustracija CFB. Izvor: [3]

6.5 Output Feedback (OFB)

Output Feedback (OFB) mod generira ključni stream koji se koristi za šifriranje čistog teksta, slično kao i CFB. Međutim, dok CFB koristi prethodni šifrirani blok kao ulaz za šifriranje sljedećeg bloka, OFB koristi prethodni izlazni blok šifriranog teksta za generiranje ključnog streama koji se koristi u XOR operaciji s čistim tekstom. Ova razlika omogućuje OFB-u bolju otpornost na pogreške u odnosu na CFB: greške u jednom bloku u OFB modu ne utječu na naknadne blokove.

- **Prednosti:** Sličan CFB modu, ali otporniji na pogreške. Ako dođe do greške u jednom bloku, ona ne utječe na sljedeće blokove.
- **Nedostaci:** Slično CFB modu, zahtijeva sekvencijalno šifriranje.

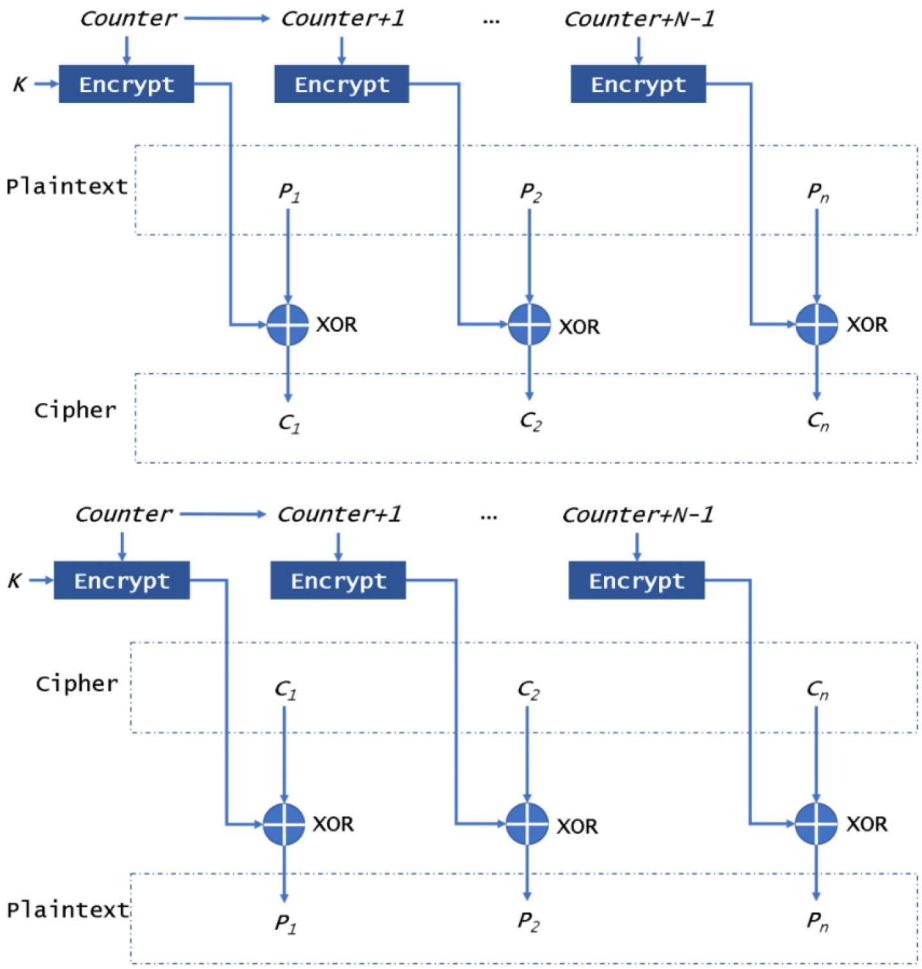


Slika 12: Ilustracija OFB. Izvor: [3]

6.6 Counter (CTR)

Counter (CTR) mod koristi brojač koji se šifrira i zatim XOR-ira s čistim tekstom za generiranje šifriranog teksta. Za razliku od ostalih modova, CTR omogućuje paralelno šifriranje i dešifriranje, što poboljšava performanse. Dok ECB obrađuje blokove neovisno, CBC zahtijeva sekvencijalno procesiranje s ovisnošću između blokova, a CFB i OFB nude samosinkronizaciju s nekim sekvencijalnim pristupom. CTR mod nudi brže performanse, ali zahtijeva pažljivo upravljanje brojačem kako bi se osigurala sigurnost.

- **Prednosti:** Omogućuje paralelno šifriranje i dešifriranje blokova, brzo generira ključni stream.
- **Nedostaci:** Potrebno je osigurati da se isti brojač ne koristi više puta s istim ključem.



Slika 13: Ilustracija CTR. Izvor: [3]

7 Napadi na Modove operacija

AES (Advanced Encryption Standard) već više od dvadeset godina predstavlja standard za šifriranje podataka i široko se koristi zbog svoje robusnosti i sigurnosti. AES-256, posebno, smatra se jednim od najsigurnijih dostupnih algoritama za šifriranje. Iako se AES smatra otpornim na različite vrste napada, sigurnost AES-a ne ovisi samo o njegovom dizajnu, već i o načinu na koji se koristi u kriptografskim sustavima. Stoga su modovi operacije AES-a od ključne važnosti za osiguranje cjelokupne sigurnosti podataka.

Međutim, svaki od ovih modova ima svoje specifične slabosti koje napadači mogu iskoristiti za dešifriranje ili manipulaciju šifriranim podacima. U ovom poglavlju istražit ćemo različite napade na najčešće korištene modove operacije AES-a, s posebnim fokusom na sljedeće napade:

- **Padding Oracle Napad** na CBC mod: Napad koji iskorištava ranjivosti u rukovanju pogrešnim paddingom kako bi postupno otkrio originalne podatke.
- **Otkrivanja obrazaca** u ECB modu: ECB mod je poznat po tome što otkriva obrasce u šifriranom tekstu, omogućavajući napadačima rekonstrukciju dijelova originalnih podataka.

Ovi napadi bit će detaljno opisani, uz popratni Python kod koji demonstrira kako se ovi napadi mogu izvesti u praksi. Važno je napomenuti da je AES, kada se pravilno koristi, još uvijek jedan od najsigurnijih dostupnih algoritama za šifriranje podataka, i to više od dvadeset godina nakon što je postao standard. Posebno AES-256, koji koristi 256-bitni ključ, pruža izuzetno visoku razinu sigurnosti i smatra se otpornim na napade čak i u kontekstu budućih kvantnih računala, što ga čini "kvantno sigurnim" u velikoj mjeri [5]. Njegova robusnost i prilagodljivost učinili su ga temeljem moderne kriptografije, a njegovo uključivanje u brojne sigurnosne protokole osigurava da će tako ostati i u nadolazećim godinama.

7.1 Electronic Codebook (ECB)

Napadi:

- **Otkrivanje obrazaca:** U ECB modu, isti blok čistog teksta uvijek se šifrira u isti blok šifriranog teksta. To omogućava napadačima da otkriju obrasce u šifriranim tekstovima, što kompromitira sigurnost.

Prevencija:

- **Korištenje sigurnijih modova:** Umjesto ECB moda, preporučuje se korištenje modova poput CBC, CFB ili CTR koji ne otkrivaju obrasce u šifriranom tekstu.

7.2 Cipher Block Chaining (CBC)

Napadi:

- **Bit-flipping napadi:** Napadač može izmijeniti pojedine bitove šifriranog teksta, što može uzrokovati predvidljive promjene u dešifriranom tekstu. To se može iskoristiti za ciljanje specifičnih dijelova podataka.

- **Padding oracle napadi:** Ako aplikacija vraća različite poruke o greškama ovisno o valjanosti paddinga, napadač može koristiti te informacije za dešifriranje šifriranog teksta.

Prevencija:

- **Provjera integriteta:** Upotreba autentifikacije poruka (npr. HMAC) može spriječiti bit-flipping napade.
- **Jednaki odgovori na greške:** Osigurati da aplikacija uvijek vraća iste poruke o greškama bez obzira na valjanost paddinga kako bi se spriječili padding oracle napadi.

7.3 Cipher Feedback (CFB)

Napadi:

- **Napadi na sinkronizaciju:** Ako dođe do gubitka ili umetanja bitova tijekom prijenosa, sinkronizacija između enkripcije i dekripcije može biti izgubljena, što uzrokuje netočan dešifrirani tekst.

Prevencija:

- **Upotreba kontrolnih zbrojeva:** Umetanje kontrolnih zbrojeva u podatke može pomoći u otkrivanju i ispravljanju grešaka tijekom prijenosa.

7.4 Output Feedback (OFB)

Napadi:

- **Bit-flipping napadi:** Kao i kod CBC moda, napadač može izmijeniti bitove šifriranog teksta što će uzrokovati predvidljive promjene u dešifriranom tekstu.

Prevencija:

- **Provjera integriteta:** Upotreba autentifikacije poruka (npr. HMAC) može spriječiti bit-flipping napade.

7.5 Counter (CTR)

Napadi:

- **Napadi putem ponovne uporabe ključeva:** Ako se isti ključ i brojač koriste više puta, napadač može izračunati XOR dvaju šifriranih tekstova i dobiti XOR dvaju čistih tekstova, što može dovesti do otkrivanja informacija.

Prevencija:

- **Jedinstveni brojači:** Osigurati da se svaki brojač koristi samo jednom s istim ključem kako bi se spriječila ponovna uporaba brojača.
- **Korištenje nonce-a:** Korištenje jedinstvenih nonce vrijednosti za svaku enkripciju kako bi se osiguralo da brojači ostanu jedinstveni.

7.6 Padding Oracle napad na CBC mod

7.6.1 Opis napada

Padding Oracle napad iskorištava način na koji sustav obrađuje padding u CBC modu. Kada se koristi CBC mod, podaci koji nisu višekratnici duljine bloka moraju biti podstavljeni prije šifriranja.

Ako aplikacija koja dešifrira poruku vraća različite poruke o grešci ovisno o tome je li podstavljanje ispravno ili ne, napadač može iskoristiti te informacije pri izvođenju napada. Takva aplikacija djeluje kao oracle koji otkriva informacije o šifriranom tekstu te omogućujući napadaču postupno otkrivanje izvornog teksta.

7.6.2 Praktični primjer: Šifriranje korisničkih podataka u CBC modu

Razmotrimo web stranicu koja šifrira korisničke podatke, kao što su sesije ili osjetljive informacije, koristeći AES u CBC modu. Ovi podaci se, primjerice, mogu pohraniti u cookie koji se koristi za autentifikaciju korisnika. Kada korisnik pošalje šifrirani podatak natrag serveru, server ga dešifrira i provjerava ispravnost podstavljanja.

Ako je podstavljanje ispravno, server vraća normalan odgovor, poput 200 OK. Međutim, ako padding nije ispravan, server može vratiti drugačiji odgovor, primjerice 500 Internal Server Error. Ovakav način rukovanja šifriranim podacima otvara mogućnost za Padding Oracle napad.

7.6.3 Kako napad funkcionira

Napadač započinje tako što izmijeni šifrirani tekst i pošalje ga serveru. Ovisno o odgovoru servera, napadač može zaključiti koje promjene u šifriranom tekstu rezultiraju ispravnim paddingom nakon dešifriranja. Ponavljanjem ovog procesa i analizom odgovora, napadač može dešifrirati cijeli šifrirani tekst, bajt po bajt.

Ovaj napad obično funkcionira na sljedeći način:

- **Odabir bloka**: Napadač odabire jedan blok šifriranog teksta i manipulira njime kako bi izazvao promjenu u paddingu dešifriranog teksta.
- **Izmjena bajta**: Napadač sustavno mijenja vrijednosti bajta u odabranom bloku i šalje modificirani šifrirani tekst serveru.
- **Analiza odgovora**: Ako server vrati odgovor koji sugerira ispravan padding, napadač zna da je pronašao ispravan bajt šifriranog teksta.
- **Postupno otkrivanje**: Proces se ponavlja za svaki bajt u bloku sve dok se ne otkrije cijeli izvorni tekst.

7.6.4 Implementacija napada

Za demonstraciju Padding Oracle napada na CBC mod, implementacija u Pythonu prikazana je u Dodatku A.1. Ova implementacija pokazuje kako napadač može dešifrirati šifrirani tekst koristeći ranjivost u provjeri ispravnosti paddinga.

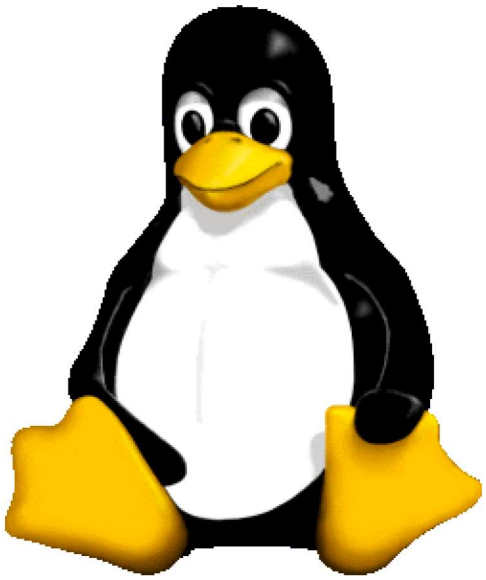
7.7 Otkrivanje obrazaca u ECB modu

7.7.1 Primjer ranjivosti: Otkrivanje obrazaca u šifriranim podacima

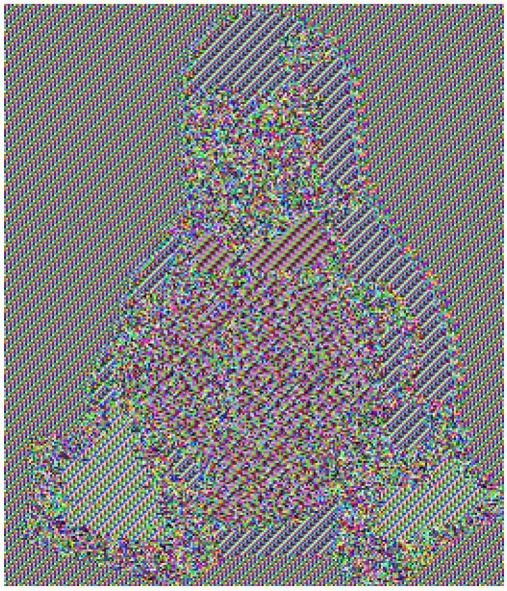
Electronic Codebook (ECB) mod je jednostavan način šifriranja gdje se svaki blok čistog teksta šifrira odvojeno, koristeći isti ključ. Iako je ovaj mod jednostavan za implementaciju, ima značajnu sigurnosnu slabost: otkrivanje obrazaca u šifriranim podacima. Ako se isti blok čistog teksta šifrira više puta, rezultirajući šifrirani blokovi bit će identični, što omogućuje napadačima da otkriju obrasce u šifriranim podacima.

7.7.2 Praktični primjer: Šifriranje slike u ECB modu

Razmotrimo primjer u kojem se šifrira slika, poput slike Tuxa (Linux logo), koristeći AES u ECB modu. Budući da svaki blok podataka (u ovom slučaju, pikseli slike) šifrira isti način, šifrirani izlaz zadržava mnoge vizualne karakteristike originalne slike, omogućujući napadaču da prepozna oblik i sadržaj slike.



Slika 14: Originalna slika Tuxa



Slika 15: Obrazci u ECB modu

Slika 16: Prikaz originalne slike i šifrirane slike u ECB modu. Izvor: [4]

7.7.3 Napad na ECB mod: Otkrivanje tajne poruke

Drugi način na koji napadač može iskoristiti ECB mod je napad byte-at-a-time, gdje se tajna poruka otkriva blok po blok. U ovom napadu, napadač koristi činjenicu da se svaki blok šifrira odvojeno kako bi sustavno pogodio svaki bajt tajne poruke.

Postupak napada može se opisati na sljedeći način:

- Napadač priprema niz podataka određene duljine i šifrira ga zajedno s tajnom porukom.
- Zatim, napadač sustavno mijenja posljednji bajt podataka dok ne pronađe odgovarajući šifrirani blok koji se podudara s šifriranim blokom tajne poruke.
- Ovaj postupak se ponavlja za svaki bajt, otkrivajući tajnu poruku jedan po jedan bajt.

7.7.4 Implementacija napada

Napad byte-at-a-time na ECB mod, zajedno s demonstracijom šifriranja slike koja zadržava vidljive obrasce, implementiran je u Pythonu, kao što je prikazano u Dodatku A.2. Ova implementacija pokazuje ranjivosti ECB moda i način na koji napadač može otkriti tajnu poruku ili vizualne obrasce u šifriranim podacima.

7.8 Zaključak

Svaki mod operacije AES-a ima svoje specifične nedostatke koje napadači mogu iskoristiti. Međutim, pravilnim sigurnosnim praksama, kao što su provjera integriteta, korištenje jedinstvenih brojača i sigurna implementacija, moguće je značajno smanjiti rizik od napada.

8 Primjene AES-a u stvarnom svijetu

AES je široko prihvaćen i koristi se u mnogim aplikacijama i industrijama za zaštitu podataka. Njegova popularnost proizlazi iz njegove sigurnosti, brzine i fleksibilnosti. U ovom poglavlju istražit će se nekoliko ključnih područja primjene AES-a.

8.1 Sigurnost podataka

AES se koristi za osiguranje povjerljivosti i integriteta podataka u različitim okruženjima. Primjene uključuju šifriranje datoteka, zaštitu baza podataka i osiguranje podataka pohranjenih na prijenosnim uređajima.

- **Šifriranje datoteka:** AES se koristi za šifriranje pojedinačnih datoteka ili cijelih direktorija na računalima i poslužiteljima, čime se spriječava neovlašten pristum osjetljivim informacijama.
- **Zaštita baza podataka:** Baze podataka često sadrže osjetljive informacije koje treba zaštititi. AES se koristi za šifriranje podataka pohranjenih u bazama podataka kako bi se osigurala povjerljivost podataka čak i ako dođe do proboja sigurnosti.
- **Zaštita prijenosnih uređaja:** AES se koristi za šifriranje podataka pohranjenih na prijenosnim uređajima poput USB stickova, vanjskih tvrdih diskova i SSD-ova kako bi se spriječio neovlašteni pristup u slučaju gubitka ili krađe uređaja.

8.2 Komunikacijski protokoli

AES se koristi u mnogim komunikacijskim protokolima za osiguranje povjerljivosti i integriteta prijenosa podataka. Primjeri uključuju HTTPS, VPN-ove i bežične mreže.

- **HTTPS:** AES je ključna komponenta HTTPS protokola koji osigurava sigurnu komunikaciju na internetu. HTTPS koristi AES za šifriranje podataka između web preglednika i poslužitelja, čime se osigurava povjerljivost i integritet podataka u prijenosu.
- **VPN-ovi:** AES se koristi u VPN (Virtual Private Network) tehnologijama za šifriranje podataka koji se prenose između udaljenih mreža ili uređaja. Ovo osigurava da podaci ostanu privatni i sigurni tijekom prijenosa preko javnih mreža.
- **Bežične mreže:** AES je integriran u standarde bežične sigurnosti poput WPA2 za osiguranje bežičnih mreža. Korištenjem AES-a, bežične mreže mogu pružiti sigurnu komunikaciju između uređaja spojenih na mrežu.

8.3 Financijske transakcije

AES se široko koristi u financijskom sektoru za zaštitu osjetljivih financijskih informacija i transakcija.

- **Šifriranje plaćanja:** AES se koristi za šifriranje podataka o kreditnim karticama i drugim osjetljivim informacijama tijekom online i offline plaćanja.
- **Bankarski sustavi:** Banke koriste AES za zaštitu podataka o računima, transakcijama i korisnicima u svojim informacijskim sustavima.

- **Bankomati:** AES se koristi u bankomatima za šifriranje PIN-ova i drugih osjetljivih podataka kako bi se spriječila krađa informacija.

8.4 Zaštita intelektualnog vlasništva

AES se koristi za zaštitu intelektualnog vlasništva u raznim industrijama, uključujući softversku i zabavnu industriju.

- **Digital Rights Management (DRM):** AES se koristi u DRM sustavima za šifriranje digitalnog sadržaja poput filmova, glazbe i e-knjiga kako bi se spriječilo neovlašteno kopiranje i distribucija.
- **Zaštita softvera:** Programeri koriste AES za zaštitu izvornog koda i binarnih datoteka softvera kako bi spriječili neovlašteni pristup i reverzno inženjerstvo.

8.5 Vladine i vojne primjene

AES je standardiziran od strane NIST-a i široko se koristi u vladinim i vojnim aplikacijama za zaštitu osjetljivih podataka.

- **Zaštita podataka:** Vladine agencije koriste AES za šifriranje povjerljivih informacija kako bi spriječile neovlašteni pristup i osigurale nacionalnu sigurnost.
- **Sigurnosni sustavi:** Vojne organizacije koriste AES za zaštitu komunikacija i podataka unutar sigurnosnih sustava i mreža.

8.6 Zaključak

AES je ključna komponenta u modernoj kriptografiji i koristi se u širokom spektru aplikacija za zaštitu podataka, komunikacija i transakcija. Njegova fleksibilnost i sigurnost čine ga idealnim izborom za mnoge različite primjene, osiguravajući povjerljivost i integritet informacija u različitim industrijama i sektorima.

A Dodatak: popis funkcija i programa

Algoritam A.1 (Sub-bytes)
U ovom dijelu prikazujemo implementaciju dvije funkcije povezane s transformacijama u AES algoritmu. Prva funkcija, `sub_bytes`, provodi ne-linearno preslikavanje poznato kao SubBytes, koje zamjenjuje svaki bajt stanja odgovarajućim bajtom iz S-kutije (eng. S-box). Inverzna funkcija, `sub_bytes_inv`, koristi inverznu S-kutiju (eng. inverse S-box) za poništavanje ove transformacije.

```
1 void sub_bytes(unsigned char *state)
2 {
3     for (int i = 0; i < AES_BLOCK_SIZE; i++)
4         state[i] = get_s_box(state[i]);
5 }
```

```
1 void sub_bytes_inv(unsigned char *state)
2 {
3     for (int i = 0; i < AES_BLOCK_SIZE; i++)
4         state[i] = get_rs_box(state[i]);
5 }
```

Algoritam A.2 (Shift-Rows)
U ovom dijelu prikazujemo implementaciju dvije funkcije povezane s transformacijama u AES algoritmu. Prva funkcija, `shift_rows`, provodi cikličko pomicanje redaka stanja u AES algoritmu. Inverzna funkcija, `shift_rows_inv`, vraća stanje na izvorni poredak redaka.

```
1  void shift_rows(unsigned char *state){
2      unsigned char tmp;
3
4      // Shift first row once to the left
5      tmp = state[4];
6      state[4] = state[5];
7      state[5] = state[6];
8      state[6] = state[7];
9      state[7] = tmp;
10
11     // Shift second row twice to the left
12     tmp = state[8];
13     state[8] = state[10];
14     state[10] = tmp;
15     tmp = state[9];
16     state[9] = state[11];
17     state[11] = tmp;
18
19     // Shift third row three times to the left
20     tmp = state[12];
21     state[12] = state[15];
22     state[15] = state[14];
23     state[14] = state[13];
24     state[13] = tmp;
25 }
```

```
1 void shift_rows_inv(unsigned char *state){
2     unsigned char tmp;
3
4     // Shift first row once to the right
```

```
5       tmp = state[7];
6       state[7] = state[6];
7       state[6] = state[5];
8       state[5] = state[4];
9       state[4] = tmp;
10
11      // Shift second row twice to the right
12      tmp = state[8];
13      state[8] = state[10];
14      state[10] = tmp;
15      tmp = state[9];
16      state[9] = state[11];
17      state[11] = tmp;
18
19      // Shift third row three times to the right
20      tmp = state[13];
21      state[13] = state[14];
22      state[14] = state[15];
23      state[15] = state[12];
24      state[12] = tmp;
25
26  }
```

Algoritam A.3 (Mix-Column)

U ovom dijelu prikazujemo implementaciju dvije funkcije povezane s transformacijama u AES algoritmu. Prva funkcija, `mix_column`, provodi miješanje stupaca u stanju koristeći Galoisovo polje. Inverzna funkcija, `mix_column_inv`, koristi inverzne operacije da poništi miješanje stupaca.

```
1  static void mix_column(unsigned char *column)
2  {
3      unsigned char tmp[4];
4      for (int i = 0; i < 4; i++)
5      {
6          tmp[i] = column[i];
7      }
8      column[0] = gf_mult(tmp[0], 2) ^
9                  gf_mult(tmp[3], 1) ^
10                 gf_mult(tmp[2], 1) ^
11                 gf_mult(tmp[1], 3);
12
13     column[1] = gf_mult(tmp[1], 2) ^
14                 gf_mult(tmp[0], 1) ^
15                 gf_mult(tmp[3], 1) ^
16                 gf_mult(tmp[2], 3);
17
18     column[2] = gf_mult(tmp[2], 2) ^
19                 gf_mult(tmp[1], 1) ^
20                 gf_mult(tmp[0], 1) ^
21                 gf_mult(tmp[3], 3);
22
23     column[3] = gf_mult(tmp[3], 2) ^
24                 gf_mult(tmp[2], 1) ^
25                 gf_mult(tmp[1], 1) ^
26                 gf_mult(tmp[0], 3);
27 }
```

```
1  static void mix_column_inv(unsigned char *column)
2  {
3      unsigned char tmp[4];
4      int i;
5      for (i = 0; i < 4; i++)
6      {
7          tmp[i] = column[i];
8      }
9      column[0] = gf_mult(tmp[0], 14) ^
10                 gf_mult(tmp[3], 9) ^
11                 gf_mult(tmp[2], 13) ^
12                 gf_mult(tmp[1], 11);
13     column[1] = gf_mult(tmp[1], 14) ^
14                 gf_mult(tmp[0], 9) ^
15                 gf_mult(tmp[3], 13) ^
16                 gf_mult(tmp[2], 11);
17     column[2] = gf_mult(tmp[2], 14) ^
18                 gf_mult(tmp[1], 9) ^
19                 gf_mult(tmp[0], 13) ^
20                 gf_mult(tmp[3], 11);
21     column[3] = gf_mult(tmp[3], 14) ^
22                 gf_mult(tmp[2], 9) ^
23                 gf_mult(tmp[1], 13) ^
```

```
24                 gf_mult(tmp[0], 11);
25 }
```

Algoritam A.4 (Cipher)
U ovom dijelu prikazujemo implementaciju dvije funkcije koje provode glavni AES algoritam šifriranja i dešifriranja. Prva funkcija, `cipher`, provodi osnovni postupak šifriranja koristeći prošireni ključ i broj rundi. Inverzna funkcija, `cipher_inv`, provodi dešifriranje koristeći iste parametre.

```
void cipher(
    unsigned char *state,
    unsigned char *expanded_key,
    int n_rounds)
{
    unsigned char round_key[AES_BLOCK_SIZE];
    create_round_key(expanded_key, round_key);
    add_round_key(state, round_key);

    for (int i = 1; i < n_rounds; i++)
    {
        create_round_key(
            expanded_key + AES_BLOCK_SIZE * i,
            round_key);

        sub_bytes(state);
        shift_rows(state);
        mix_columns(state);
        add_round_key(state, round_key);
    }
    create_round_key(
        expanded_key + AES_BLOCK_SIZE * n_rounds,
        round_key);
    sub_bytes(state);
    shift_rows(state);
    add_round_key(state, round_key);
}
```

```
void cipher_inv(
    unsigned char *state,
    unsigned char *expanded_key,
    int n_rounds)
{

    unsigned char round_key[AES_BLOCK_SIZE];
    create_round_key(
        expanded_key + AES_BLOCK_SIZE * n_rounds,
        round_key);
    add_round_key(state, round_key);

    for (int i = n_rounds - 1; i > 0; i--)
    {
        create_round_key(
            expanded_key + AES_BLOCK_SIZE * i,
            round_key);
        shift_rows_inv(state);
        sub_bytes_inv(state);
        add_round_key(state, round_key);
        mix_columns_inv(state);
    }
    create_round_key(expanded_key, round_key);
    shift_rows_inv(state);
```

```
25      sub_bytes_inv(state);
26      add_round_key(state, round_key);
27  }
```

Algoritam A.5 (Key-Expansion)
U ovom dijelu prikazujemo implementaciju funkcije `key_expansion` koja se koristi u AES algoritmu za generiranje proširenog ključa iz početnog ključa. Prošireni ključ koristi se u svakom krugu šifriranja i dešifriranja.:

```
1   void key_expansion(
2       unsigned char *expanded_key,
3       unsigned char *key,
4       enum aes_key_size size)
5   {
6       int n_rounds = get_nr(size);
7       int n_k = get_nk(size);
8
9       for (int i = 0; i <= n_k; i++)
10      {
11          expanded_key[4 * i + 0] = key[4 * i + 0];
12          expanded_key[4 * i + 1] = key[4 * i + 1];
13          expanded_key[4 * i + 2] = key[4 * i + 2];
14          expanded_key[4 * i + 3] = key[4 * i + 3];
15      }
16
17      unsigned char tmp[4] = {0};
18      for (int i = n_k; i < 4 * (n_rounds + 1); i++)
19      {
20          for (int k = 0; k < 4; k++)
21          {
22              tmp[k] = expanded_key[(i - 1) * 4 + k];
23          }
24
25          if (i % n_k == 0)
26          {
27              key_schedule(tmp, i / n_k);
28          }
29          else if (n_k > 6 && i % n_k == 4)
30          {
31              for (int k = 0; k < 4; k++)
32              {
33                  tmp[k] = get_s_box(tmp[k]);
34              }
35          }
36          for (int k = 0; k < 4; k++)
37          {
38              expanded_key[4 * i + k] =
39                  expanded_key[(i - n_k) * 4 + k] ^ tmp[k];
40          }
41      }
42  }
```

Algoritam A.6 (PKCS#7 padding)
U ovom dijelu prikazujemo implementaciju funkcije `create_PKCS7_buffer` koja se koristi u AES algoritmu za generiranje paddinga. Padding se koristi u CBC i EBC modovima AES-a.

```
1  static void create_PKCS7_buffer(
2      unsigned char *buffer_to_pad,
3      int buffer_to_pad_size,
4      int bytes_to_append,
5      unsigned char *output_buffer)
6  {
7      int padded_size = input_buffer_size + bytes_to_append;
8      int pad_length = bytes_to_append;
9
10     memcpy(output_buffer, input_buffer, input_buffer_size);
11
12     for (int i = input_buffer_size; i < padded_size; i++)
13     {
14         output_buffer[i] = (unsigned char)pad_length;
15     }
16 }
```

A.1 Python Implementacija Padding Oracle Napada na CBC Mod

```
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad
from Crypto.Random import get_random_bytes
import base64

BLOCK_SIZE = 16 # AES block size in bytes

key = get_random_bytes(BLOCK_SIZE) # Generate a random AES key

def encrypt(plaintext):
    """
    Encrypt the plaintext using AES in CBC mode with a random IV.
    """
    # Generate a random IV
    iv = get_random_bytes(BLOCK_SIZE)
    # Create a new AES cipher
    cipher = AES.new(key, AES.MODE_CBC, iv)
    # Pad and encrypt the plaintext
    ciphertext = cipher.encrypt(pad(plaintext, BLOCK_SIZE))
    # Prepend IV to ciphertext
    return iv + ciphertext

def is_padding_ok(ciphertext):
    """
    Check if the decrypted ciphertext has valid padding.
    """
    # Extract the IV
    iv = ciphertext[:BLOCK_SIZE]
    # Extract the ciphertext
    ciphertext = ciphertext[BLOCK_SIZE:]
    # Create a new AES cipher
    cipher = AES.new(key, AES.MODE_CBC, iv)
    try:
        # Decrypt and check padding
        unpad(cipher.decrypt(ciphertext), BLOCK_SIZE)
        return True # Padding is valid
    except ValueError:
        return False # Padding is invalid

def attack(ciphertext):
    """
    Perform the padding oracle attack to decrypt the ciphertext.
    """
    guessed_clear = b''
    # Split the data into blocks
    split_string = lambda x, n: [x[i:i+n] for i in range(0, len(x), n)]
    blocks = split_string(ciphertext, BLOCK_SIZE)

    # Process each pair of blocks
    for block_n in range(len(blocks) - 1, 0, -1):
        # Pair of blocks to attack
        spliced_ciphertext = blocks[block_n - 1] + blocks[block_n]
        # Placeholder for decrypted bytes
        decoded_bytes = b'?' * BLOCK_SIZE

        # Process each byte in the block
```

```
57          for byte in range(BLOCK_SIZE - 1, -1, -1):
58              new_pad_len = BLOCK_SIZE - byte
59              hacked_ciphertext_tail = b''
60
61              for padder_index in range(1, new_pad_len):
62                  hacked_ciphertext_tail +=
63                      bytearray([new_pad_len ^ decoded_bytes[byte + padder_index]])
64
65              for i in range(256): # Try all possible byte values
66                  attack_str = bytearray([(i ^ spliced_ciphertext[byte])])
67                  hacked_ciphertext = spliced_ciphertext[:byte] +
68                                      attack_str + hacked_ciphertext_tail +
69                                      spliced_ciphertext[byte + 1 + new_pad_len - 1:]
70                  # Check if padding is valid
71                  if is_padding_ok(hacked_ciphertext):
72                      test_correctness = hacked_ciphertext[:byte - 1] +
73                      bytearray([(1 ^ hacked_ciphertext[byte])]) +
74                      hacked_ciphertext[byte:]
75
76                      # Check if padding is correct
77                      if not is_padding_ok(test_correctness):
78                          continue
79
80                      decoded_bytes = decoded_bytes[:byte] + bytearray([hacked_ciphertext[
                            byte] ^ new_pad_len]) + decoded_bytes[byte + 1:]
81                      guessed_clear = bytearray([i ^ new_pad_len]) + guessed_clear
82                      break
83
84      return guessed_clear[:-guessed_clear[-1]] # Remove padding
85
86  if __name__ == '__main__':
87      messages = [b'Attack␣at␣dawn',
88                  b'',
89                  b'Filip',
90                  b"In␣symmetric␣cryptography,␣the␣padding␣oracle␣attack␣can␣be␣applied␣to␣
                        the␣CBC␣mode␣of␣operation,␣where␣the␣\"oracle\"␣(usually␣a␣server)␣
                        leaks␣data␣about␣whether␣the␣padding␣of␣an␣encrypted␣message␣is␣correct
                        ␣or␣not.␣Such␣data␣can␣allow␣attackers␣to␣decrypt␣(and␣sometimes␣
                        encrypt)␣messages␣through␣the␣oracle␣using␣the␣oracle's␣key,␣without␣
                        knowing␣the␣encryption␣key"
91                  ]
92
93      for msg in messages:
94          ciphertext = encrypt(msg)
95          cracked_ct = attack(ciphertext)
96          print('Success:', cracked_ct)
```

A.2 Python Implementacija Napada na ECB Mod

```
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad
import os

block_size = 16 # AES block size in bytes

secret_key = os.urandom(16) # Generate a random 128-bit AES key

def aes_encrypt(data):
    """
    Encrypt the data using AES in ECB mode with the secret key.
    """
    cipher = AES.new(secret_key, AES.MODE_ECB) # Create a new AES cipher in ECB mode
    padded_data = pad(data, AES.block_size) # Pad the data to ensure it's a multiple of
        the block size
    return cipher.encrypt(padded_data) # Encrypt the padded data

def encrypt_with_secret(data):
    """
    Encrypt the given data concatenated with the secret message from 'flag.txt' using AES
        in ECB mode.
    """
    with open("flag.txt", "rb") as f:
        secret_message = f.read().strip() # Read and strip the contents of 'flag.txt'
    return aes_encrypt(data + secret_message) # Concatenate the input data with the
        secret message and encrypt

def ecb_attack():
    """
    Perform an ECB byte-at-a-time attack to recover the secret message.
    """
    discovered_data = b"" # Initialize the discovered data as an empty byte string
    while True:
        block_index = len(discovered_data) // block_size # Determine the current block
            index
        offset = block_size - 1 - (len(discovered_data) % block_size) # Calculate the
            number of padding bytes needed
        input_data = b"A" * offset # Prepare the padding data to align the blocks

        # Encrypt the padding data with the secret to get the target block
        encrypted_block = encrypt_with_secret(input_data)[:block_size * (block_index + 1)]

        for i in range(256):
            # Prepare the test data with the discovered data and the current byte guess
            test_data = input_data + discovered_data + bytes([i])
            # Encrypt the test data with the secret
            test_encrypted = encrypt_with_secret(test_data)[:block_size * (block_index +
                1)]
            if test_encrypted == encrypted_block:
                # If the encrypted block matches the target block, the guessed byte is
                    correct
                discovered_data += bytes([i])
                break
        else:
            # If no match is found, the entire secret message has been discovered
            break
```

```
50
51      return discovered_data
52
53  # Perform the ECB byte-at-a-time attack and recover the secret message
54  recovered_message = ecb_attack()
55  print("Recovered Message:", recovered_message)
```

Literatura

[1] Daemen, J., Rijmen, V. *AES Proposal: Rijndael.* National Institute of Standards and Technology (NIST), 2001. Dostupno na: `https://csrc.nist.gov/csrc/media/projects/cryptographic-standards-and-guidelines/documents/aes-development/rijndael-ammended.pdf`

[2] AES NIST. Dostupno na: `https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197-upd1.pdf` (16. studeni 2001.)

[3] AES Modovi operacija. Dostupno na: `https://www.highgo.ca/2019/08/08/the-difference-in-five-modes-in-the-aes-encryption-algorithm/` (16. studeni 2001.)

[4] Tux slike. Dostupno na: `https://github.com/FilipVuk123/AESinC/tree/main/hackingAES/ECB/TUX`

[5] Markus Grassl, Brandon Langenberg, Martin Roetteler, and Rainer Steinwandt, *Applying Grover's Algorithm to AES: Quantum Resource Estimates*, Post-Quantum Cryptography, 2016.

Sažetak

Ovaj rad istražuje implementaciju i primjenu Naprednog standarda za šifriranje (AES) u programskom jeziku C, s posebnim fokusom na pet najčešćih načina rada: ECB, CBC, CFB, OFB i CTR. AES, simetrična blok šifra standardizirana od strane NIST-a, ključna je za osiguranje povjerljivosti podataka. Studija se bavi teoretskim osnovama AES-a, uključujući njegovu aritmetiku konačnog polja, te prikazuje praktične detalje implementacije za svaki način rada. Razmatraju se i sigurnosni aspekti svakog načina rada, čime se demonstrira robusnost i svestranost AES-a u različitim kriptografskim scenarijima. Važno je napomenuti da je, kada je ispravno implementiran, AES pokazao otpornost na kvantne napade, što ga čini preferiranim izborom u mnogim primjenama kao što su sigurne komunikacije, pohrana podataka i financijske transakcije.

Ključne riječi

Ključne riječi: AES, Napredni standard za šifriranje, programski jezik C, kriptografski načini rada, sigurnost podataka, otpornost na kvantne napade, aritmetika konačnog polja, ECB, CBC, CFB, OFB, CTR, sigurne komunikacije, pohrana podataka, financijske transakcije.

Schur decomposition and its applications

Abstract

This thesis explores the implementation and application of the Advanced Encryption Standard (AES) in the C programming language, focusing on the five most common modes of operation: ECB, CBC, CFB, OFB, and CTR. AES, a symmetric block cipher standardized by NIST, is vital for ensuring data confidentiality. The study delves into the theoretical underpinnings of AES, including its finite field arithmetic, and presents practical implementation details for each mode. Security considerations for each mode is discussed, demonstrating AES's robustness and versatility in various cryptographic scenarios. Notably, when implemented correctly, AES has shown resistance to quantum attacks, making it a preferred choice in many applications such as secure communications, data storage, and financial transactions.

Keywords

AES, Advanced Encryption Standard, C programming, cryptographic modes of operation, data security, quantum resistance, finite field arithmetic, ECB, CBC, CFB, OFB, CTR, secure communications, data storage, financial transactions.

Životopis

Rođen sam 03.11.1999. godine u Osijeku gdje sam pohađao osnovnu i srednju školu. Po završetku I. Gimnazije Osijek, 2018. godine upisao sam preddiplomski studij na Odjelu za matematiku, smjer matematike i računarstva. Nakon završetka preddiplomskog studija nastavljam na diplomski studij smijer matematike i računarstva. Pri završetku diplomskog studija zaposlen sam u tvrtki Orqa gdje radim na nisko latentnim sustavima za prijenos video podataka na bespilotnim letjelicama i vozilima.