Matematička pozadina AES kriptosustava

Doko, Dubravka

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni studij

MATEMATIČKA POZADINA AES KRIPTOSUSTAVA

Diplomski rad

Dubravka Doko

Osijek, 2022.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 01.09.2022.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime Pristupnika:	Dubravka Doko
Studij, smjer:	Diplomski sveučilišni studij Elektrotehnika, smjer Komunikacije i informatika'
Mat. br. Pristupnika, godina upisa:	D-1287, 13.10.2020.
OIB studenta:	16736594137
Mentor:	Doc.dr.sc. Anita Katić
Sumentor:	Izv. prof. dr. sc. Krešimir Grgić
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	Doc.dr.sc. Tomislav Rudec
Član Povjerenstva 1:	Doc.dr.sc. Anita Katić
Član Povjerenstva 2:	Doc. dr. sc. Višnja Križanović
Naslov diplomskog rada:	Matematička pozadina AES kriptosustava
Znanstvena grana diplomskog rada:	**Telekomunikacije i informatika (zn. polje elektrotehnika)**
Zadatak diplomskog rada:	Objasniti matematičke pojmove koji se koriste u AES kriptosustavu. Na temelju matematičke razrade napraviti vlastiti kod u nekom od viših programskih jezika te ga usporediti s gotovim implementacijama. Sumentor: izv.prof.dr.sc. Krešimir Grgić
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	01.09.2022.

Potvrda mentora o predaji konačne verzije rada:	*Mentor elektronički potpisao predaju konačne verzije.*
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

IZJAVA O ORIGINALNOSTI RADA

Osijek, 07.09.2022.

Ime i prezime studenta:	Dubravka Doko
Studij:	Diplomski sveučilišni studij Elektrotehnika, smjer Komunikacije i informatika'
Mat. br. studenta, godina upisa:	D-1287, 13.10.2020.
Turnitin podudaranje [%]:	13

Ovom izjavom izjavljujem da je rad pod nazivom**: Matematička pozadina AES kriptosustava**

izrađen pod vodstvom mentora Doc.dr.sc. Anita Katić

i sumentora Izv. prof. dr. sc. Krešimir Grgić

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija. Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

IZJAVA

o odobrenju za pohranu i objavu ocjenskog rada

kojom ja __Dubravka Doko_____, OIB: __16736594137_____, student/ica Fakulteta elektrotehnike, računarstva i informacijskih tehnologija Osijek na studiju _Diplomski sveučilišni studij Elektrotehnika, smjer Komunikacije i informatika' _, kao autor/ica ocjenskog rada pod naslovom: _Matematička pozadina AES kriptosustava**,**

dajem odobrenje da se, bez naknade, trajno pohrani moj ocjenski rad u javno dostupnom digitalnom repozitoriju ustanove Fakulteta elektrotehnike, računarstva i informacijskih tehnologija Osijek i Sveučilišta te u javnoj internetskoj bazi radova Nacionalne i sveučilišne knjižnice u Zagrebu, sukladno obvezi iz odredbe članka 83. stavka 11. *Zakona o znanstvenoj djelatnosti i visokom obrazovanju* (NN 123/03, 198/03, 105/04, 174/04, 02/07, 46/07, 45/09, 63/11, 94/13, 139/13, 101/14, 60/15).
Potvrđujem da je za pohranu dostavljena završna verzija obranjenog i dovršenog ocjenskog rada. Ovom izjavom, kao autor/ica ocjenskog rada dajem odobrenje i da se moj ocjenski rad, bez naknade, trajno javno objavi i besplatno učini dostupnim:

a) široj javnosti
b) studentima/icama i djelatnicima/ama ustanove
c) široj javnosti, ali nakon proteka 6 / 12 / 24 mjeseci (zaokružite odgovarajući broj mjeseci).

**U slučaju potrebe dodatnog ograničavanja pristupa Vašem ocjenskom radu, podnosi se obrazloženi zahtjev nadležnom tijelu Ustanove.*

__Osijek, 07.09.2022._____ _____

(mjesto i datum) (vlastoručni potpis studenta/ice)

Sadržaj

1. UVOD

Internetska komunikacija igra važnu ulogu u prijenosu velike količine podataka u različitim područjima. Neki se podaci mogu prenositi kanalom koji je nesiguran, a da toga nismo ni svjesni. Iz toga razloga privatni i javni sektori koriste različite tehnike i metode za zaštitu osjetljivih podataka od zlonamjernih osoba jer je sigurnost elektroničkih podataka ključni problem. Kriptografija je jedna od najznačajnijih i najpopularnijih tehnika za zaštitu podataka od napadača pomoću dva vitalna procesa, a to su šifriranje i dešifriranje. Šifriranje je proces kodiranja podataka kako bi se spriječilo napadače da lako čitaju izvorne podatke. Ovom fazom se pretvaraju izvorni podaci u nečitljiv format poznat kao šifrirani tekst. Sljedeći proces koji mora izvršiti ovlaštena osoba jest suprotan od procesa šifriranja, a naziva se dešifriranje. To je proces pretvaranja šifriranog teksta u običan tekst bez propuštanja riječi u izvornom tekstu. Za izvođenje ovih procesa kriptografija se oslanja na matematičke izračune zajedno s nekim zamjenama i permutacijama sa ili bez ključa.

Postoji niz algoritama koji su dostupni za šifriranje i dešifriranje osjetljivih podataka koji se obično dijele u tri vrste. Prva je simetrična kriptografija koja se koristi istim ključem za šifriranje i dešifriranje podataka. Drugi je asimetrična kriptografija. Ova vrsta kriptografije oslanja se na dva različita ključa za šifriranje i dešifriranje. Naposljetku, kriptografska funkcija raspršivanja koja ne koristi ključ, umjesto ključa miješa podatke [2].

Simetrični ključ je mnogo učinkovitiji i brži od asimetričnog. Neki od uobičajenih simetričnih algoritama su Advance EncryptionStandard (AES), Blowfish, Simplified DataEncryption Standard (S-DES) i 3DES. Glavna svrha ovog rada jest pružiti detaljne informacije o Advanced Encryption Standard (AES) algoritmu za šifriranje i dešifriranje podataka.

1.1. Zadatak diplomskog rada

Zadatak ovog diplomskog rada je objasniti matematičke pojmove koji se koriste u AES kriptosustavu. Na temelju matematičke razrade napraviti vlastiti kod u nekom od viših programskih jezika te ga usporediti s gotovim implementacijama.

1.2. Organizacija rada

Ovaj rad je organiziran na način da je u odjeljku 2 opisan kratki pregled i povijest nastanka AES algoritma, kao i usporedba sa DES-om. Matematička pozadina AES kriptosustava predstavljena je u odjeljku 3. U odjeljku 4 navedeni su slojevi AES algoritma kao i matematička primjena. Napadi na AES algoritam opisani su u odjeljku 5, a u odjeljku 6 predstavljeno je programsko rješenje za AES blok šifru sa veličinom ključa od 128-bita razvijenu u Java programskom jeziku. U odjeljku 7 dan je zaključak ovoga rada.

2. POVIJEST AES-A

2.1. Što je DES?

Data Encryption Standard (DES) je kriptografski algoritam koji je izabran kao standardni federalni algoritam u Sjedinjenim Američkim Državama 1976. godine. Dizajnirala ga je američka tvrtka IBM. DES je blok šifra, što znači da ne šifrira jedan po jedan bit, već istovremeno šifrira blok podataka. DES grupira poruku otvorenog teksta u 64-bitne blokove tijekom enkripcije. Blokovi se kodiraju koristeći ključ duljine 56 bita u 64-bitni šifrirani tekst korištenjem supstitucije i transpozicije. DES se danas smatra nesigurnim algoritmom zbog kratkog ključa koji se može razbiti napadima „grubom silom“ koje ćemo objasniti u nastavku ili metodama koje zahtijevaju manje računalnih ciklusa. Danas je velikim dijelom uporabu DES-a zamijenio *Advanced Encryption Standard* (AES).[1]

Natječaj za novi standard AES

National Institute of Standards and Technology (NIST) je 1999. godine donio odluku da se DES treba koristit samo za naslijeđene sustave i da se umjesto njega treba koristiti trostruki DES (3DES). 3DES je algoritam koji tri puta primjenjuje DES na informacije koje se šifriraju. Iako se 3DES odupire napadima grube sile s današnjom tehnologijom, postoje drugi problemi koji se javljaju. Nije učinkovit u pogledu implementacije softvera, 3DES je tri puta sporiji od DES-a. Drugi problem je relativno kratka veličina bloka od 64 bita, kao i duljina ključa koja je prekratka. Sva ova razmatranja dovela su NIST do zaključka da je potrebna potpuno nova blok šifra kao zamjena za DES.[2]

Godine 1997. NIST je raspisao natječaj za novi napredni standard šifriranja AES. Za razliku od razvoja DES-a, odabir algoritma za AES bio je otvoren proces kojim je upravljao NIST. Unutar poziva za dostavu prijedloga, sljedeći uvjeti bili su obavezni za sve prijedloge:

- simetričan blokovni kriptosustav,
- blok šifra sa veličinom bloka od 128 bita,
- moraju biti podržane tri duljine ključa: 128, 192 i 256 bita,
- sigurnost u odnosu na druge dostavljene algoritme,
- efikasnost i primjenjivost na software i hardware.[2]

U lipnju 1998. godine je zaključen natječaj i od 21 pristigle prijave njih 15 je zadovoljilo kriterije. U sljedeća tri kruga evaluacije, NIST i međunarodna znanstvena zajednica raspravljali su o prednostima i nedostacima dostavljenih prijedloga te su suzili broj potencijalnih kandidata.[3]

U kolovozu 1999. NIST je odabrao pet algoritama za opsežniju analizu:

- MARS kojeg je podnio veliki tim iz IBM Researcha,
- RC6 dostavljen od RSA Security,
- Rijndael kojeg su podnijela dva belgijska kriptografa, Joan Daemen i Vincent Rijmen,
- Serpent podnijeli su ga Ross Anderson, Eli Biham i Lars Knudsen,
- Twofish koji je podnio veliki tim istraživača iz Counterpane Internet Security, uključujući poznatog kriptografa Brucea Schneiera.[3]

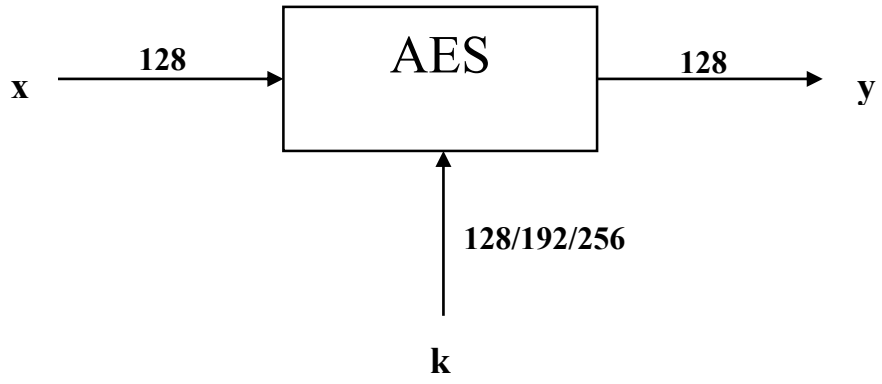
Implementacije svega gore navedenog opsežno su testirane u Američkom nacionalnom institutu za standarde (ANSI), C i Java jezicima za:

- brzina i pouzdanost u procesima šifriranja i dešifriranja,
- vrijeme postavljanja ključa i algoritma,
- otpornost na razne napade, kako u hardverskim tako i u softverskim sustavima.

Dana 2. listopada 2000. NIST je objavio da je odabrao Rijndaela kao AES. U studenom 2001. godine AES je službeno odobren kao američki savezni standard. Slijede očekivanja da će AES u sljedećih nekoliko desetljeća biti dominantni algoritam za mnoge komercijalne aplikacije. Također je izvanredno da je 2003. godine američka Agencija za nacionalnu sigurnost (NSA) objavila da dopušta AES-u šifriranje povjerljivih dokumenata do razine SECRET za sve duljine ključa i do razine TOP SECRET za duljine ključa od 192 ili 256 bita. Prije tog datuma, za šifriranje povjerljivih dokumenata koristili su se samo nejavni algoritmi.[3]

2.2. Pregled AES algoritma

Blok šifra koja je pobijedila na natječaju Rijndael sadrži blokove koji variraju između 128, 192 i 256 bita. Međutim, AES standard zahtijeva samo veličinu bloka od 128 bita. Stoga je samo Rijndael s duljinom bloka od 128 bita poznat kao AES algoritam. [4]



Slika 2.1 *Ulazno/izlazni parametri AES-a*

Rijndael mora podržavati tri duljine ključa 128, 192 i 256 bita. Broj unutarnjih krugova šifre ovisi o duljini ključa te je prikazano u Tablici 2.1.

Tablica 2.1 *Duljine ključeva i broj rundi za AES*

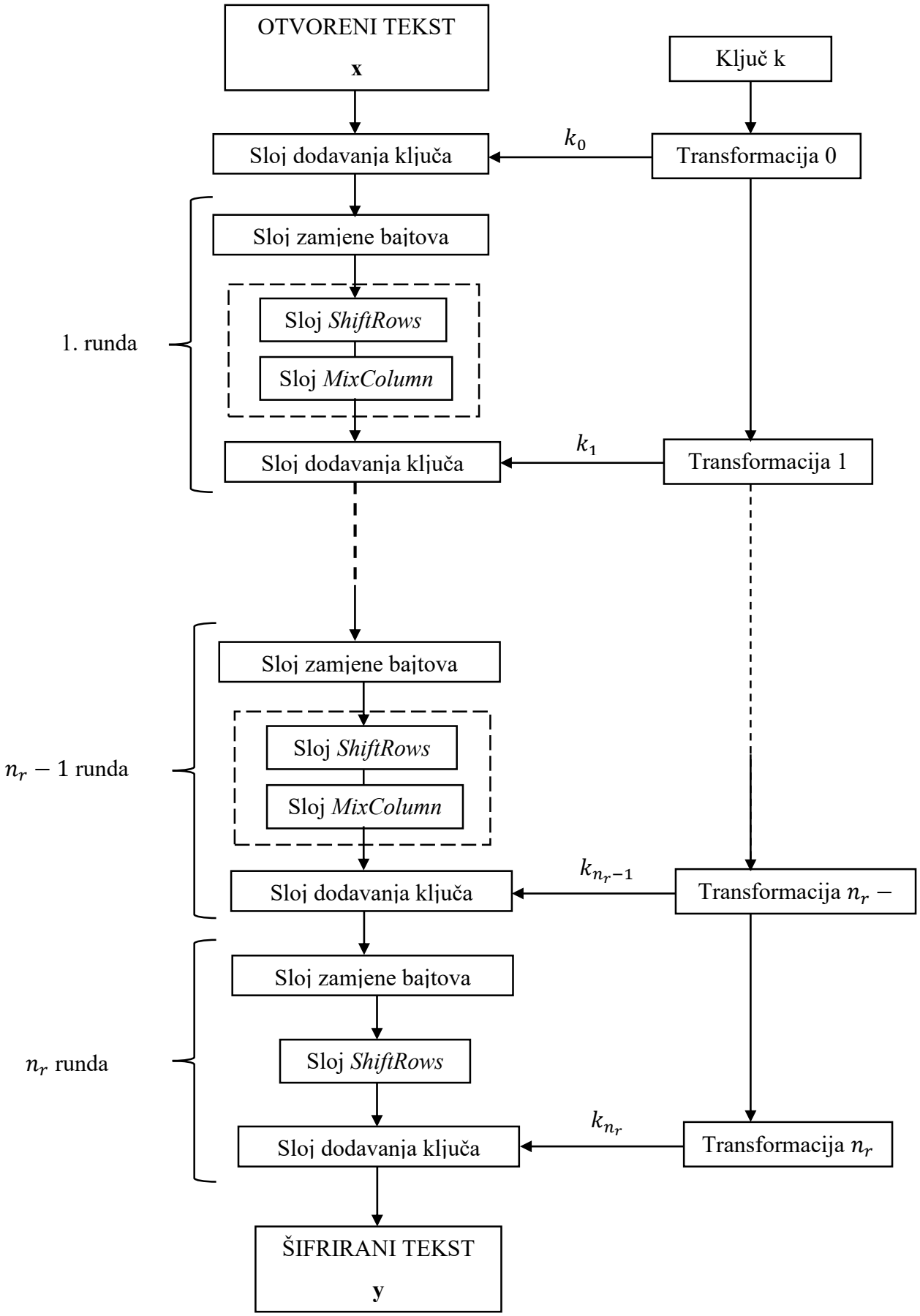
DULJINA KLJUČA	**BROJ RUNDI n_r**
128 bita	10
192 bita	12
256 bita	14

Za razliku do DES algoritma, AES nije zasnovan na Feistelovoj strukturi gdje se ne šifrira cijeli blok po iteraciji već kao kod npr. DES-a $^{64}/_{2} = 32$ bita se šifrira u jednom krugu. S druge strane, AES šifrira svih 128 bita po jednoj iteraciji. To je jedan od razloga zašto ima relativno mali broj krugova.[4]

AES se sastoji od slojeva. Svaki sloj manipulira sa svih 128 bita podatkovnog puta, gdje se taj put podataka naziva stanje algoritma. Razlikujemo tri vrste slojeva:

- sloj dodavanja ključa (eng. *Key Addition layer*),
- sloj zamjene bajtova (eng. *Byte Substitution layer* (S-Box)),
- difuzijski sloj (eng. *Diffusion layer*).[4]

Svaki se krug, osim prvog, sastoji od sva tri sloja kao što je prikazano na Slici 2.2. Otvoreni tekst označen je sa x, šifrirani tekst sa y, a broj rundi sa n_r. Kao što možemo vidjeti sa Slike 2.2 posljednja runda ne koristi transformaciju *MixColumn*, što shemu šifriranja i dešifriranja čini simetričnom.



Slika 2.2 *Blok dijagram AES šifriranja*

Slijedi kratki opis slojeva.

Sloj dodavanja ključa (eng. *Key Addition layer*)- Imamo n-bitni ključ k koji je poznat kao glavni ključ. Primjenjujemo algoritam proširenja ključa i generiramo potreban broj okruglih potključeva. U ovom sloju koristimo bitwise XOR operaciju za miješanje okruglog potključa s ulaznim podacima.

Sloj zamjene bajtova (eng. *Byte Substitution layer* (S-Box))- Ovaj sloj koristi nelinearnu funkciju poznatu kao kutija za zamjenu (S-box). Svaki element stanja se nelinearno transformira korištenjem preglednih tablica s posebnim matematičkim svojstvima. Kažemo da ovaj sloj unosi zbrku u podatke, odnosno osigurava da se promjene u pojedinačnim stanjima bita šire brzo preko podatkovnog puta.

Difuzijski sloj (eng. *Diffusion layer*)- Također ga nazivamo i permutacijski sloj. Ovaj sloj se koristi za brzu difuziju ulaznih bitova. Sastoji se od dva podsloja koji izvode linearne operacije:

- sloj ***ShiftRows*** permutira podatke na razini bajtova,
- sloj ***MixColumn*** je matrična operacija koja kombinira (miješa) blokove od četiri bajta.[3]

Prije nego što detaljno opišemo unutarnje slojeve, moramo se osvrnuti na matematičku pozadinu AES kriptosustava. Matematička pozadina nam je vrlo bitna jer su izračuni za Galoisova polja potrebni za sve operacije unutar AES slojeva.

3. MATEMATIČKA POZADINA AES-a

Aritmetika Galoisova polja se koristi u većini slojeva kod AES-a, a naročito u *S-Box* i *MixColumn* sloju. Kako bi dublje razradili unutrašnjost AES-a, u ovom odjeljku dani su pojmovi i tvrdnje iz algebarskih struktura koje vežemo uz konačna polja, odnosno nama vrlo važnom Galoisovom polju (2^8).

3.1. Konačno polje

Konačno polje, koje se ponekad naziva i Galoisovo polje, predstavlja skup s konačnim brojem elemenata u kojem možemo zbrajati, oduzimati, množiti i invertirati. Prije uvođenja definicije polja, potreban nam je koncept jednostavnije algebarske strukture, grupe.[5]

3.1.1. Grupa

Definicija 3.1.1.1 Grupa

Neprazan skup elemenata G zajedno s binarnom operacijom · (množenja) predstavlja grupu ako vrijede sljedeća svojstva:

1. *operacija · je zatvorena, odnosno za svaki* $a, b \in G$ *vrijedi da je* $a \cdot b = c \in G$,
2. *asocijativnost:* $a \cdot (b \cdot c) = (a \cdot b) \cdot c$ *za sve* $a, b, c \in G$,
3. *postoji element* $1 \in G$, *koji se naziva neutralni element (ili element identiteta), takav da je* $a \cdot 1 = 1 \cdot a = a$ *za svaki* $a \in G$ *(također se element identiteta umjesto jedinice zna označavati sa* e*),*
4. *za svaki* $a \in G$ *postoji element* $a^{-1} \in G$, *kojeg nazivamo inverzni element od* a, *takav da je* $a \cdot a^{-1} = a^{-1} \cdot a = 1$,
5. *grupa G je abelova ili komutativna ako je* $a \cdot b = b \cdot a$ *za sve* $a, b \in G$.

Grupu sa pripadajućom binarnom operacijom zapisujemo kao uređeni par $(G, \cdot)$ ili pak kao G ukoliko je binarna operacija poznata iz konteksta. [5]

Također možemo promatrati skupove na kojima su definirane dvije binarne operacije koje nazivamo zbrajanje i množenje.

3.1.2. Prsten

Definicija 3.1.2.1. Prsten

Neprazan skup R zajedno s dvije binarne operacije + i · (zbrajanje i množenje) predstavlja prsten ako su za svaki $a, b, c \in R$ *zadovoljena sljedeća svojstva:*

1. *(R, +) je Abelova grupa,*
2. $(R, \cdot)$ *je polugrupa (vrijedi zatvorenost i asocijativnost),*
3. *distributivnost:* $a \cdot (b + c) = a \cdot b + a \cdot c$, $(a + b) \cdot c = a \cdot c + b \cdot c$,

Neutralni element iz Abelove grupe (R, +) označavamo 0 i nazivamo ga nula prstena, dok inverzni element označavamo sa -a i nazivamo ga suprotni element.

Primjer 3.1.2.2. Ukoliko uzmemo u obzir skup cijelih brojeva od 0 do m-1 zajedno s operacijama zbrajanja i množenja dobivamo sljedeću matematičku konstrukciju:

Cjelobrojni prsten $\mathbb{Z}_m$ je skup $\mathbb{Z}_m = \{0, 1, 2, \dots, m-1\}$ i dvije operacije " $+$ " i " $\cdot$ " koje su definirane sa:

a. $a + b \equiv c \ (mod\ m), (a, b, c \in \mathbb{Z}_m)$
b. $a \cdot b \equiv d \ (mod\ m), (a, b, d \in \mathbb{Z}_m)$. [3]

Pri tome " $\equiv$ " predstavlja oznaku za relaciju kongruencije na skupu $\mathbb{Z}_m$ te za dva cijela broja a, b i prirodni broj m vrijedi da je a kongruentan b modulo m (pišemo: $a \equiv b \ (mod\ m)$) ako i samo ako brojevi a i b pri dijeljenju s m daju isti ostatak.[7] Relacija kongruencije će u sljedećem poglavlju biti detaljnije objašnjena.

U nastavku ovoga primjera prikazano je da vrijede svojstva koja moraju vrijediti da bi neka struktura bila prsten:

- za prsten kažemo da je zatvoren, možemo zbrajati i množiti bilo koja dva broja, a rezultat će uvijek biti u prstenu,
- asocijativnost: $a + (b + c) = (a + b) + c$ i $a \cdot (b \cdot c) = (a \cdot b) \cdot c$ za sve $a, b, c \in \mathbb{Z}_m$,
- distributivnost: $a \cdot (b + c) = a \cdot b + a \cdot c$ za sve $a, b, c \in \mathbb{Z}_m$, možemo reći da je prsten $\mathbb{Z}_m$ skup cijelih brojeva $\{0, 1, 2, \dots, m-1\}$ u kojima možemo zbrajati, oduzimati, množiti, a ponekad i dijeliti,
- postoji neutralni element 0 s obzirom na zbrajanje, odnosno vrijedi za svaki element $a \in \mathbb{Z}_m$ vrijedi da je $a + 0 \equiv a \ (mod\ m)$,
- za bilo koji element a u prstenu uvijek postoji negativni element $-a$ takav da je $a + (-a) \equiv 0 \ (mod\ m)$, odnosno aditivni inverz uvijek postoji.
- postoji neutralni element 1 s obzirom na množenje, odnosno vrijedi za svaki element $a \in \mathbb{Z}_m$ vrijedi da je $a \cdot 1 \equiv a \ (mod\ m)$.

- multiplikativni inverz postoji samo za neke elemente, neka je $a \in \mathbb{Z}$ definiran tako da vrijedi $a \cdot a^{-1} = 1 \ (mod\ m)$. [6]

Za pronalazak inverza nekog elementa obično se koristi Euklidov algoritam. Međutim postoji jednostavniji način da se utvrdi postoji li inverz za dati element a ili ne. Ova metoda pronalaska inverza temelji se na pronalasku najvećeg zajedničkog djelitelja (u nastavku nzd), tj. najvećeg cijelog boja koji dijeli oba broja a i m. Kažemo da element $a \in \mathbb{Z}$ ima multiplikativni inverz a^{-1} ako i samo ako je $nzd(a, m) = 1$. Ukoliko je zadovoljeno $nzd(a, m) = 1$, onda se za a i m kaže da su relevantno prosti. [3]

Primjer 3.1.2.3. Pogledajmo postoji li multiplikativni inverz od 15 u $\mathbb{Z}_{26}$

S obzirom da je $nzd(15, 26) = 1$ mora postojati inverz.

Primjer 3.1.2.4. Pogledajmo postoji li multiplikativni inverz od 14 u $\mathbb{Z}_{26}$.

Promotrivši odnos $nzd(14, 26) = 2 \neq 1$ možemo zaključiti da multiplikativni inverz od 14 ne postoji u $\mathbb{Z}_{26}$.

Iz ovoga možemo vidjeti i zaključiti da je prsten $\mathbb{Z}_m$, a samim time i cjelobrojna aritmetika s operacijom po modulu od temeljne važnosti za modernu kriptografiju. Stoga nam slijedi objašnjenje modularne aritmetike.

3.1.3 Modularna aritmetika

Bilo da se radi o simetričnim ili asimetričnim šiframa, gotovo se svi kripto algoritmi temelje na aritmetici konačnog broja elemenata. Većina uobičajenih skupova brojeva kao što su skup prirodnih brojeva ili skup realnih brojeva su beskonačni. Jednostavan način izvođenja aritmetike u konačnom skupu cijelih brojeva predstavlja modularna aritmetika koju uvodimo u nastavku.

Najprije ćemo promotriti primjer konačnog skupa cijelih brojeva iz svakodnevnog života koji slijedi u nastavku.

Primjer 3.1.3.1. Uzimamo u obzir brojevne oznake na satu.

Ako nastavimo dodavati jedan sat, dobivamo:

$$1h, 2h, 3h, 4h, \dots, 11h, 12h, 1h, 2h, \dots, 11h, 12h, 1h, 2h, \dots$$

Možemo vidjeti da iako neprestano dodajemo jedan sat, uvijek imamo isti set brojeva.[3]

Nadalje, promotrimo primjer općeg načina postupanja s aritmetikom u takvim konačnim skupovima.

Primjer 3.1.3.2. Promatramo skup od 9 brojeva:

$$\{0, 1, 2, 3, 4, 5, 6, 7, 8\}$$

Možemo raditi redovitu aritmetiku sve dok je rezultat manji od 9. Na primjer:

$$2 \cdot 3 = 6$$

$$4 + 4 = 8$$

Nadalje, gledamo što će se dogoditi ako je rezultat veći od 9, kao na primjer $8 + 4$. Slijedi pravilo koje kaže da se najprije izvrši redovna cjelobrojna aritmetika te se zatim rezultat podijeli sa 9. U sljedećem koraku uzimamo samo ostatak dijeljenja, a ne izvorni rezultat. Budući da je $8 + 4 = 12$, a ${}^{12}/_{9} = 1$ pri čemu je ostatak cjelobrojnog dijeljenja jednak 3, pišemo:

$$8 + 4 \equiv 3 \ (mod\ 9)$$

Slijedi točna definicija modulo operacije:

Definicija 3.1.3.3. Modulo operacija

Neka je $a, r, m \in \mathbb{Z}$ *(gdje je* $\mathbb{Z}$ *skup svih cijelih brojeva) i* $m > 0$. *Pišemo*

$$a \equiv r \ (mod\ m)$$

ako m dijeli $a - r$.

m *se naziva modul, a* r *nazivamo ostatak.*

Kako bismo sve četiri osnovne aritmetičke operacije (zbrajanje, oduzimanje, množenje i dijeljenje) imali u jednoj strukturi, potreban nam je skup koji sadrži aditivnu i multiplikativnu grupu. To je ono što nazivamo poljem.[8]

3.1.4. Polje

Definicija 3.1.4.1. Polje

Polje F predstavlja skup elemenata za koje vrijedi:

1. *aditivnu grupu tvore svi elementi F s operacijom grupe "+" i neutralnim elementom 0,*
2. *multiplikativnu grupu tvore svi elementi F osim 0 s operacijom grupe " · " i neutralnim elementom 1,*
3. *distributivnost* $a \cdot (b + c) = a \cdot b + a \cdot c$ *za sve* $a, b, c \in F$.

Odnosno kažemo da je $(F, +, \cdot)$ *polje.*[8]

Primjer 3.1.4.2. Skup realnih brojeva $\mathbb{R}$ je polje s neutralnim elementom 0 za aditivnu grupu i elementom 1 za multiplikativnu skupinu. Svaki realni broj a ima aditivni inverz $-a$, a svaki element različit od 0 ima multiplikativni inverz $1/a$.

U većini slučajeva kada govorimo o kriptografiji zanimaju nas polja s konačnim brojem elemenata odnosno konačna polja. Broj elemenata sadržan u nekom poju naziva se kardinalnost polja, te nam je važan sljedeći teorem.

Teorem 3.1.4.3. *Polje s redom* m *postoji samo ako je* m *potencija prostog broja odnosno* $m = p^n$ *gdje je* n *pozitivan cijeli broj, a* p *prosti cijeli broj.* p *se naziva i karakteristika konačnog polja.*

Teorem 3.1.4.3. implicira da postoje konačna polja s npr. 11 elemenata ili 81 elemenata ($3^4 = 81$), ili pak 256 elemenata ($2^8 = 256$). Isto tako ne postoji konačno polje s npr. 12 elemenata, jer 12 nije potencija prostog broja. [3]

3.2. Osnovna polja

Osnovni primjer konačnih polja su polja prostog reda gdje je $n = 1$ odnosno $m = p$. Elementi polja $GF(p)$ mogu biti predstavljeni nizom cijelih brojeva $0, 1, \dots, p - 1$. Polje $GF(p)$ koristi dvije operacije: modularno zbrajanje cijelog broja i množenje cijelog broja po modulu p.

Teorem 3.2.1. *Neka je p prost broj. Cjelobrojni prsten $\mathbb{Z}_p$ označava se kao $GF(p)$ i naziva se prostim poljem ili Galoisovim poljem s prostim brojem elemenata. Svi elementi $GF(p)$ koji su različiti od nule imaju inverz. Aritmetika u $GF(p)$ se vrši po modulu p.*[9]

Ako uzmemo u obzir cjelobrojni prsten $\mathbb{Z}_m$ uveden u primjeru 3.1.2.2. tj. cijeli brojevi s modularnim zbrajanjem i množenjem, znači da ukoliko je m prost broj, $\mathbb{Z}_p$ ne predstavlja samo prsten nego je on i konačno polje $(m = p)$.

Da bismo mogli računati u prostom polju moramo slijediti pravila za cjelobrojne prstenove odnosno pravila zbrajanja i množenja koji se vrše po modulu p, aditivni inverz za bilo koji element je zadan sa $a + (-a) \equiv 0\ (mod\ p)$, te multiplikativni inverz bilo kojeg elementa različitog od nule definiran je kao $a \cdot a^{-1} \equiv 1$.[9] U nastavku slijedi primjer jednostavnog polja.

Primjer 3.2.2. Najmanje konačno polje je $GF(2) = \{0, 1\}$. Računanje se jednostavno odvija po modulu 2 te daje sljedeće tablice zbrajanja i množenja:

+	**0**	**1**
0	0	1
1	1	0

·	**0**	**1**
0	0	0
1	0	1

Polje $GF(2)$ je vrlo važno za AES. Iz gornjeg primjera možemo vidjeti da je $GF(2)$ množenje ekvivalentno logičkoj AND operaciji, a zbrajanje po modulu 2 je ekvivalentno logičkoj XOR operaciji.

Primjer 3.2.3. Proučimo konačno polje $GF(5) = \{0, 1, 2, 3, 4\}$. Tablice u nastavku opisuju kako zbrajati i množiti bilo koja dva elementa, te kako odrediti aditivni i multiplikativni inverz od elemenata polja. Vrlo je važno istaknuti da je neutralni element za operaciju zbrajanja 0, dok je neutralni element za množenje 1. Pomoću ovih tablica možemo izvesti sve izračune u ovom polju.[8]

+	**0**	**1**	**2**	**3**	**4**
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

·	**0**	**1**	**2**	**3**	**4**
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

Aditivni inverz

$-0 = 0$

$-1 = 4$

$-2 = 3$

$-3 = 2$

$-4 = 1$

Multiplikativni inverz

$0^{-1} = ne\ postoji$

$1^{-1} = 1$

$2^{-1} = 3$

$3^{-1} = 2$

$4^{-1} = 4$

3.3. Proširena polja 2^m

Kod AES-a koristimo konačno polje koje sadrži 256 elemenata i označavamo ga kao $GF(2^8)$. Polje $GF(2^8)$ je odabrano iz razloga što se svaki element polja može prikazati jednim bajtom odnosno s 8 bitova. Za izvedbu *S-Box* i *MixColumn* operacija koje ćemo objasniti u nastavku ovoga diplomskog rada, AES tretira svaki bajt podataka kao element polja $GF(2^8)$ i manipulira podacima koristeći računske operacije u tom konačnom polju.

Međutim, ukoliko red konačnog polja nije prost, a 2^8 očigledno nije prost, u tom slučaju se operacije zbrajanja i množenja ne mogu predstaviti zbrajanjem i množenjem cijelih brojeva po modulu 2^8. Polja 2^m u kojima je $m > 1$ nazivamo proširena polja. Kako bi se mogli baviti tim poljima potrebna nam je drugačija notacija za elemente polja kao i različita pravila za izvođenje proračuna s elementima u tom polju. U nastavku ovoga diplomskog rada vidjet ćemo da su elementi proširenog polja predstavljeni polinomom. Također će biti opisane operacije koje primjenjujemo u tom polju.[10]

U proširenim poljima $GF(2^m)$ elementi nisu predstavljeni cijelim brojevima već su prikazani kao polinomi s koeficijentima iz gore navedenog $GF(2)$ polja. Maksimalni stupanj koji polinomi mogu imati je $m-1$, tako da za svaki element polja imamo ukupno m koeficijenata na raspolaganju. U polju koje AES koristi $GF(2^8)$ je svaki element $A \epsilon GF(2^8)$ predstavljen kao:

$$A(x) = a_7 \cdot x^7 + \cdots + a_1 \cdot x + a_0, \quad a_i = GF(2) = \{0, 1\}.$$

Potrebno je uočiti da postoji točno 256 odnosno 2^8 takvih polinoma, a da skup tih 256 polinoma predstavlja konačno polje $GF(2^8)$. Također svaki taj polinom se može jednostavno pohraniti u digitalnom obliku kao 8-bitni vektor:

$$A = (a_7, a_6, a_5, a_4, a_3, a_2, a_1, a_0).$$

Odnosno možemo zaključiti da ne moramo pohranjivati faktore kao npr. x^7 , već iz same pozicije bitova je jasno kojoj potenciji x^i pripada svaki koeficijent.[10]

3.3.1. Zbrajanje i oduzimanje u $GF(2^m)$

U ključnom sloju AES-a koristi se operacija zbrajanja. U ovom odlomku slijedi objašnjenje operacija zbrajanja i oduzimanja u proširenim poljima $GF(2^m)$. Ove operacije se jednostavno postižu izvođenjem standardnog polinomskog zbrajanja i oduzimanja, odnosno zbrajamo ili oduzimamo koeficijente uz jednake potencije varijable x. Također zbrajanje ili oduzimanje koeficijenata se vrši u temeljnom $GF(2)$ polju.[3]

Definicija 3.3.1.1. Zbrajanja i oduzimanje proširenih polja $GF(2^m)$

Neka su $A(x), B(x) \epsilon GF(2^m)$. Zbroj ta dva elementa se izračunava prema:

$$C(x) = A(x) + B(x) = \sum_{i=0}^{m-1} c_i \cdot x^i, \; c_i \equiv a_i + b_i \; (mod\; 2)$$

A razlika se računa kao:

$$C(x) = A(x) - B(x) = \sum_{i=0}^{m-1} c_i \cdot x^i, \; c_i \equiv a_i - b_i \; (mod\; 2)$$

Važno je primijetiti da se koeficijenti zbrajaju ili oduzimaju po modulu 2, a te dvije operacije su zapravo iste. Kao što smo prije spomenuli zbrajanje po modulu 2 je jednako logičkoj XOR operaciji. Slijedi primjer u polju $GF(2^8)$ koje se koristi u AES-u.[3]

Primjer 3.3.1.2. Zbroj dvaju polinoma $A(x) = x^7 + x^6 + x^4 + 1$ i $B(x) = x^4 + x^2 + 1$, pri čemu je $C(x) = A(x) + B(x)$.

$$=> C(x) = x^7 + x^6 + x^2$$

Imajmo na umu da bismo dobili isti rezultat kao i za zbroj ukoliko bismo računali razliku $A(x) - B(x)$ iz gornjeg primjera.

3.3.2. Množenje u $GF(2^m)$

Množenje u $GF(2^8)$ je osnovna operacije u *MixColumn* sloju AES-a. Prilikom množenja najprije se dva elementa koja su predstavljena polinomima konačnog polja $GF(2^m)$ množe korištenjem standardnog pravila množenja polinoma.

Primjer 3.3.2.1. Neka su $A(x), B(x) \epsilon GF(2^m)$, gdje je $A(x) = a_{m-1} \cdot x^{m-1} + \cdots + a_0$ i $B(x) = b_{m-1} \cdot x^{m-1} + \cdots + b_0$. Produkt ta dva elementa se izračunava prema:

$$A(x) \cdot B(x) = (a_{m-1} \cdot x^{m-1} + \cdots + a_0) \cdot (b_{m-1} \cdot x^{m-1} + \cdots + b_0)$$

$$C'(x) = c'_{2m-2} \cdot x^{2m-2} + \cdots + c'_0$$

gdje je:

$$c'_0 = a_0 \cdot b_0 \ (mod\ 2)$$

$$c'_1 = a_0 \cdot b_1 + \ a_1 \cdot b_0 \ (mod\ 2)$$

...

$$c'_{2m-2} = a_{m-1} \cdot b_{m-1} \ \ (mod\ 2)$$

Svi koeficijenti $a_i, b_i \ i \ c_i$ su elementi polja $GF(2)$, te se sav proračun izvodi u $GF(2)$. Općenito, produkt polinoma $C(x)$ ima veći stupanj od $m - 1$, pa se mora smanjiti. Pristup je sličan kao kod množenja u prostim poljima. Odnosno pomnožimo dva cijela broja, zatim rezultat podijelimo sa prostim brojem i na kraju gledamo ostatak dijeljenja. U konačnim poljima ćemo rezultat množenja podijeliti s polinomom, a zatim promatrati ostatak toga dijeljenja polinoma. U tu svrhu su nam potrebni ireducibilni polinomi. Ireducibilni polinom su polinomi koje možemo usporediti s prostim brojevima, gdje su im jedini faktori broj 1 i oni sami. [3]

Definicija 3.3.2.2. Množenje proširenih polja $GF(2^m)$

Neka su $A(x), B(x) \epsilon GF(2^m)$ i neka je $P(x) = \sum_{i=0}^{m} p_i \cdot x^i, \ p_i \epsilon GF(2)$ ireducibilni polinom. Množenje dva elementa $A(x)$ i $B(x)$ se izvodi kao:

$$C(x) \equiv A(x) \cdot B(x)\ (mod\ P(x)).$$

Pri čemu $C(x)$ predstavlja ostatak pri dijeljenju umnoška polinoma $A(x) \cdot B(x)$ s ireducibilnim polinomom $P(x)$. Ireducibilni polinom $P(x)$ je m-tog stupnja s koeficijentima iz $GF(2)$. Važno je shvatiti da nisu svi polinomi ireducibilni. Na primjer, polinom $x^4 + x^3 + x + 1$ je reducibilan jer je $x^4 + x^3 + x + 1 = (x^2 + x + 1) \cdot (x^2 + 1)$, te se samim time taj polinom se ne može koristiti za konstruiranje polja proširenja $GF(2^4)$. Kao dio AES specifikacije navedeno je da je njegov ireducibilni polinom $P(x) = x^8 + x^4 + x^3 + x + 1$.[4]

Primjer 3.3.2.3. Pomnožimo dva polinoma $A(x) = x^3 + x^2 + 1$ i $B(x) = x^2 + x$ u polju $GF(2^4)$. Ireducibilni polinom ovog konačnog polja je $P(x) = x^4 + x + 1$.

Pri čemu se umnožak polinoma u polju računa kao: $C'(x) = A(x) \cdot B(x) = x^5 + x^3 + x^2 + x$.

Sljedeći korak je reducirati $C'(x)$ standardnom metodom dijeljenja polinoma. Ponekad je jednostavnije reducirati svaki od vodećih članova pojedinačno =>

$$x^4 = 1 \cdot P(x) + (x + 1)$$

$$x^4 \equiv x + 1\ (mod\ P(x))$$

$$x^5 \equiv x^2 + x\ (mod\ P(x))$$

Nakon što smo reducirali polinom potrebno je zamijeniti x^5 sa dobivenim izrazom u $C'(x)$.

$$C(x) \equiv x^5 + x^3 + x^2 + x\ (mod\ P(x))$$

$$C(x) = (x^2 + x\ ) + (x^3 + x^2 + x) = x^3$$

$$A(x) \cdot B(x) = x^3$$

U ovom slučaju ne smijemo brkati množenje u $GF(2^m)$ sa cjelobrojnim množenje, pogotovo kada se bavimo softverskom implementacijom Galoisova polja. Polinomi, odnosno elementi polja se obično u računalima pohranjuju kao bitni vektori. Ukoliko množenje iz prethodnog primjera promotrimo sa strane izvođenja operacije na razini bita tada slijedi:

$$C(x) = A(x) \cdot B(x) = (x^3 + x^2 + 1) \cdot (x^2 + x) = x^3$$

$$C(x) = A(x) \cdot B(x) = (1\ 1\ 0\ 1) \cdot (0\ 1\ 1\ 0) = (1\ 0\ 0\ 0)$$

Ukoliko ove nizove bita pretvorimo iz binarnog u dekadski sustav, možemo vidjeti da ovo izračunavanje nije identično cjelobrojnoj aritmetici =>

$$(1\ 1\ 0\ 1)_2 = 13_{10}$$

$$(0\ 1\ 1\ 0)_2 = 6_{10}$$

pri čemu bi rezultat bio

$$(1\ 1\ 0\ 1\ 0\ 1\ 1\ 0)_2 = 78_{10}$$

što očito nije rezultat množenja konačnog polja. Stoga, iako elemente polja možemo predstaviti kao cjelobrojne tipove podataka, ne možemo koristiti cjelobrojnu aritmetiku.[4]

3.3.3. Inverz u $GF(2^m)$

Inverzija u $GF(2^8)$ je bitna operacija transformacije zamjene bajtova koja se odvija u *S-Box*-ovima kod AES-a. Za dano konačno polje $GF(2^m)$ i odgovarajući ireducibilni polinom $P(x)$, inverz A^{-1} ne-nul elementa $A \in GF(2^m)$ definiran je kao:

$$A^{-1} \cdot A(x) = 1\ (mod\ P(x))$$

Za manja polja, koja u praksi sadrže 2^{16} ili manje elemenata, često se koriste tablice pretraživanja koje sadrže unaprijed izračunate inverze svih elemenata polja. Vrijednosti koje se koriste unutar S-Box-a AES-a prikazuje sljedeća tablica:[3]

Tablica 3.1 *Inverzna tablica u GF(2^8) za bajtove xy koji se koriste unutar AES S-Boxa*

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

Tablica 3.1 sadrži sve inverze u $GF(2^8)$ po modulu $P(x) = x^8 + x^4 + x^3 + x + 1$ u heksadecimalnom zapisu. Iz tablice možemo iščitati da je poseban slučaj za element polja 0, za koji inverz ne postoji. Međutim, za AES *S-Box* potrebna je zamjenska tablica koja je definirana za svaku moguću ulaznu vrijednost. Stoga je *S-Box* definiran tako da se ulazna vrijednost 0 preslikava u izlaznu vrijednost 0.[3]

Primjer 3.3.3.1. Iz tablice 3.1 iščitavamo inverz od $x^7 + x^6 + x$.

$$x^7 + x^6 + x = (1\ 1\ 0\ 0\ 0\ 0\ 1\ 0)_2 = (C2)_{16} = (2F)$$

Dakle, najprije smo polinom prikazali pomoću 8-bitnog vektora, zatim smo pretvorili iz dekadskog u heksadekadski sustav, te na kraju očitali vrijednost iz tablice. Vrijednost iz tablice se očitava na način da se gleda element C u retku i element 2 u stupcu pri čemu dobivamo rezultat 2F što bi povratkom u dekadski sustav bilo $(0\ 0\ 1\ 0\ 1\ 1\ 1\ 1)_{10} \Rightarrow x^5 + x^3 + x^2 + x + 1$.

To se može provjeriti množenjem:

$(x^7 + x^6 + x) \cdot (x^8 + x^4 + x^3 + x + 1) \equiv 1(mod\ P(x))$.

Gornja tablica ne predstavlja samu S-Box, već je ona nešto složenija te će biti opisana u odjeljku 4.1.

4. UNUTARNJA POZADINA AES-a

Kako bismo razumjeli na koji se način podaci kreću kroz AES strukturu, najprije moramo zamisliti nekakvo stanje A. Stanje A predstavlja 128-bitni put podataka koje možemo prikazati pomoću 16 bajtova $(A_0, A_1, \dots, A_{15})$ raspoređenih u kvadratnu matricu veličine $4x4$.

Tablica 4.1 *Matrica stanja*

A_o	A_4	A_8	A_{12}
A_1	A_5	A_9	A_{13}
A_2	A_6	A_{10}	A_{14}
A_3	A_7	A_{11}	A_{15}

Kao što ćemo vidjeti u nastavku, AES radi sa elementima koji se nalaze u stupcima ili redcima matrice trenutnog stanja. Vrlo slično su raspoređeni bajtovi ključa u matrici koja se sastoji od četiri retka dok broj stupaca ovisi o samoj duljini ključa. Tako 128-bitni ključ ima četiri stupca, 192-bitni ključ šest stupaca i 256-bitni ključ osam stupaca. [2]

Tablica 4.2 *Matrični prikaz ključeva*

128-bitni ključ

k_o	k_4	k_8	k_{12}
k_1	k_5	k_9	k_{13}
k_2	k_6	k_{10}	k_{14}
k_3	k_7	k_{11}	k_{15}

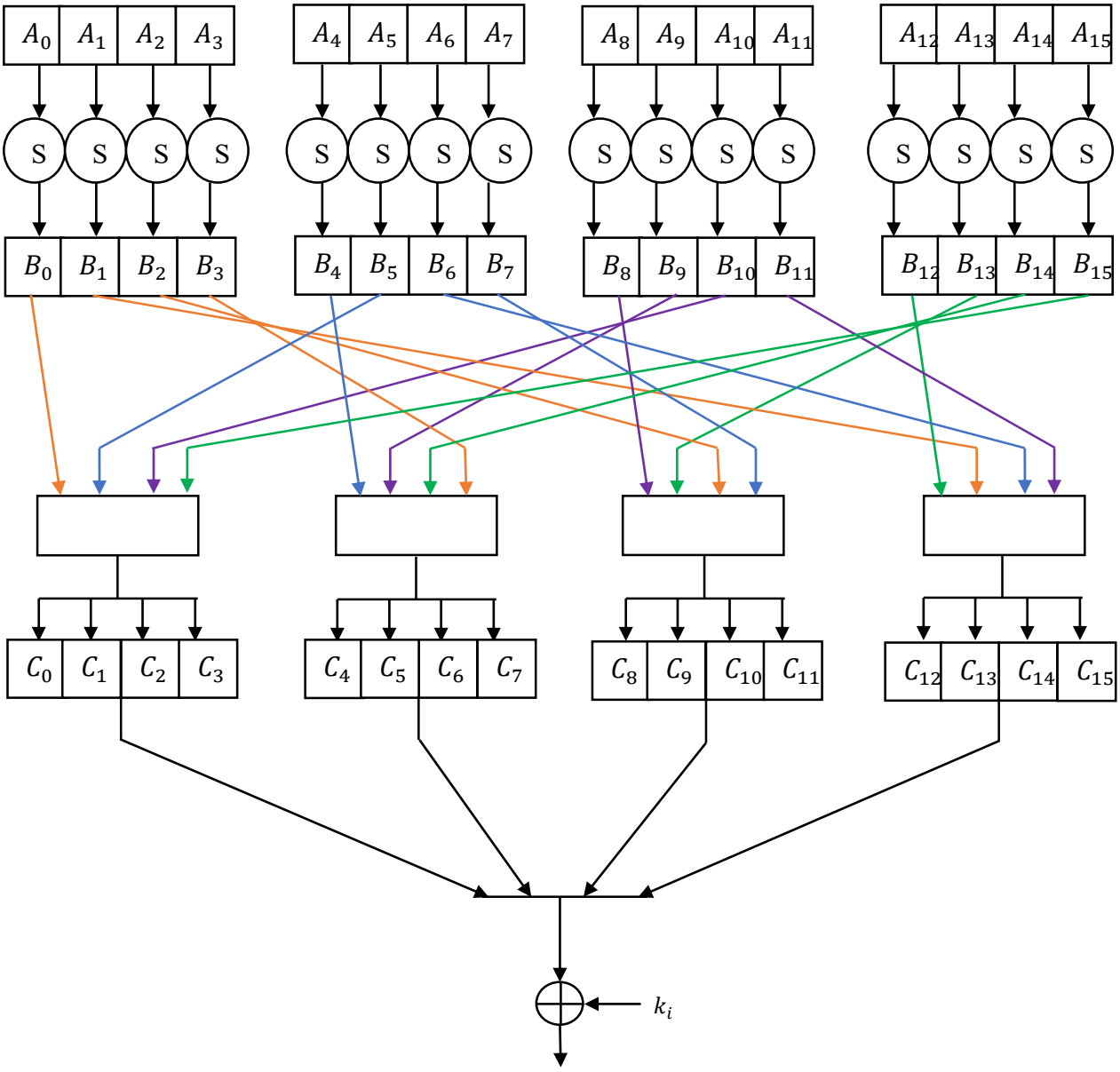
192-bitni ključ

k_o	k_4	k_8	k_{12}	k_{16}	k_{20}
k_1	k_5	k_9	k_{13}	k_{17}	k_{21}
k_2	k_6	k_{10}	k_{14}	k_{18}	k_{22}
k_3	k_7	k_{11}	k_{15}	k_{19}	k_{23}

256-bitni ključ

k_o	k_4	k_8	k_{12}	k_{16}	k_{20}	k_{24}	k_{28}
k_1	k_5	k_9	k_{13}	k_{17}	k_{21}	k_{25}	k_{29}
k_2	k_6	k_{10}	k_{14}	k_{18}	k_{22}	k_{26}	k_{30}
k_3	k_7	k_{11}	k_{15}	k_{19}	k_{23}	k_{27}	k_{31}

Prije opisa svakog od slojeva prikazan je graf jedne AES runde. Najprije imamo 16-bajtni ulaz $(A_0, A_1, \dots, A_{15})$ koji se unosi po bajtu u S-Box. Izlaz $(B_0, B_1, \dots, B_{15})$ se permutira po bajtu u sloju ShiftRows te se miješa MixColumn transformacijom $c(x)$. Na kraju se vrši XOR operacija između 128-bitnog potključa i srednjeg rezultata.[3]



Slika 4.1 *Prikaz jedne AES runde*

Važno je još jednom napomenuti da je AES bajt orijentirana šifra. S druge strane treba spomenuti da prethodnik DES koristi permutaciju bitova te se može smatrati da ima bit-orijentiranu strukturu. Sada slijedi opis svih slojeva.

4.1. Sloj zamjene bajtova (eng. *Byte Substitution layer* (S-Box))

Sloj zamjene bajtova je prva operacija u svakoj rundi AES-a. Sloj zamjene bajtova se može promatrati kao red od 16 paralelnih S-kutija, svaka kutija se sastoji od 8 ulaznih i izlaznih bitova. U sloju je svaki bajt stanja A_i zamijenjen sa drugim bajtom B_i, odnosno $S(A_i) = B_i$.

S-kutija je matrica bajtova dimenzije $16x16$ (Tablica 4.3.). Element bloka matrice $4x4$ jest bajt, odnosno 8 bita gdje prva četiri bita predstavljaju x os, a druga četiri bita y os u S-kutiji.

S-kutija predstavlja jedini nelinearni dio AES-a pa vrijedi da je $ByteSub(A) + ByteSub(B) \neq ByteSub(A + B)$. Zamjena S-kutijom je bijektivno preslikavanje, odnosno svaki od $2^8 = 256$ mogućih ulaznih elemenata odgovara točno jednom izlaznom elementu. To nam omogućuje da jedinstveno preokrenemo S-kutiju što je potrebno u postupku dešifriranja. U softverskim implementacijama S-kutija je obično realizirana kao prikazana Tablica 4.3 s fiksnim unosima.[8]

Tablica 4.3 *Prikaz S-Box-a*

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

Primjer 4.1.1. Pretpostavimo da je ulazni bajt u S-kutiju $A_i = (D1)_{hex}$, tada će zamijenjena vrijednost biti $S((D1)_{hex}) = (3E)_{hex}$.

Vrlo bitna stvar u enkripciji jest manipulacija bitovima, tako da se ova zamjena može opisati kao $S(11010001) = (00111110)$.

Iako je S-kutija bijektivna, ona nema fiksne točke, odnosno ne postoje ulazne vrijednosti A_i takve da su $S(A_i) = A_i$. Možemo vidjeti iz Tablice 4.3 da ni nulti ulaz nije fiksna točka $S(00000000) = (01100011)$.

Primjer 4.1.2. Pretpostavimo da je ulaz u sloj zamjene bajtova $(D1, D1, \dots, D1)_{hex}$. Izlazno stanje će tada biti $(3E, 3E, \dots, 3E)_{hex}$.

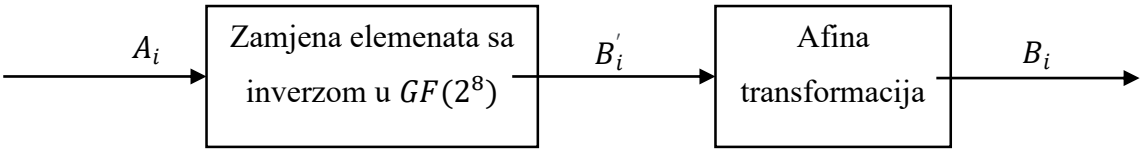
Primjer 4.1.3. Slijedi primjer korištenja transformacije za sljedeću matricu:

A3	19	7C	F0
7D	AE	B4	99
3A	91	C7	0F
D7	CC	4B	73

Nakon zamjene elemenata pomoću S-kutije naša matrica poprima sljedeće vrijednosti:

0A	D4	10	8C
FF	E4	8D	EE
80	81	C6	76
0E	4B	B3	8F

Slijedi matematički opis S-kutije s obzirom da imaju snažnu algebarsku strukturu. S-kutija AES-a se može promatrati kao matematička transformacija u dva koraka (Slika 4.2).



Slika 4.2 *Dvije operacije unutar AES S-Boxa koje izračunavju funkciju* $B_i = S(A_i)$

Prvi dio se odnosi na zamjenu svakog elementa AES-bloka sa svojim inverzom u Galoisovom polju odnosno $GF(2^8)$ čija je matematička pozadina predstavljena u odjeljku 3.2. Za svaki ulazni element A_i se izračunava inverz $B_i' = {A_i}^{-1}$, gdje su A_i i B_i' elementi polja $GF(2^8)$ s fiksnim ireducibilnim polinomom $P(x) = x^8 + x^4 + x^3 + x + 1$. Tablica sa svim inverzima prikazana je u Tablici 3.1. Imajte na umu da inverzni element nule ne postoji. Međutim, za AES je definirano da se nulti element $A_i = 0$ preslikava sam na sebe.

U drugom dijelu zamjene, svaki bajt B_i' se množi s konstantnom bit-matricom nakon čega slijedi dodavanje konstantnog 8-bitnog vektora. Ova operacija opisana je sa:

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{pmatrix} \equiv \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} mod\ 2$$

Važno je napomenuti da je $B' = (b'_7, b'_6, \dots, b'_0)$ bit-ni vektorski prikaz $B'_i = {A_i}^{-1}$. Ovaj korak naziva se afina transformacija. Pogledajmo primjer kako funkcioniraju izračuni.[3]

Primjer 4.1.4. Pretpostavimo da je ulaz u S-kutiju $A_i = (11010001)_2 = (D1)_{hex}$. Iz Tablice 3.1 iščitavamo da je inverz $B'_i = {A_i}^{-1} = (07)_{hex} = (00000111)_2$.

Sada primjenjujemo vektor bita B'_i kao ulaz za afinu transformaciju. Bit najmanjeg značaja (eng. *least significant bit* LSB) b'_0 od B'_i se nalazi na krajnjoj desnoj poziciji.

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{pmatrix} \equiv \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} mod\ 2 = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

$$B_i = (0\ 0\ 1\ 1\ 1\ 1\ 1\ 0) = (3E)_{hex}$$

Isti takav rezultat smo dobili u Primjeru 4.1.1. koji je dobiven iz S-kutije pomoću Tablice 4.3.

Tablicu 4.3 možemo dobiti izračunavanjem oba koraka od svi 256 mogućih ulaznih elemenata S-kutije. U većini implementacija AES-a, posebno u gotovo svim softverskim realizacijama, izlazi S-kutije nisu eksplicitno izračunate kao što je ovdje prikazano, već se koriste tablice pretraživanja poput Tablice 4.3. Međutim, za hardverske implementacije ponekad je korisno realizirati S-kutije kao digitalne sklopove koji zapravo izračunavaju inverz iza kojeg slijedi afina transformacija.

Prednost korištenja inverzije u $GF(2^8)$ kao temeljne funkcije sloja zamjene bajtova je u tome što osigurava visok stupanj nelinearnosti, pri čemu pruža optimalnu zaštitu od nekih najjačih poznatih analitičkih napada. S druge strane afina transformacija "uništava" algebarsku strukturu

Galoisova polja, što je pak potrebno za sprječavanje napada koji bi iskoristili inverziju konačnog polja.[3]

4.2. Difuzijski sloj (eng. *Diffusion layer*)

Difuzijski sloj se sastoji od dva podsloja, transformacije ShiftRows i transformacije MixColumn. Difuzija predstavlja širenje utjecaja pojedinih bitova na cijelo stanje. Za razliku od S-kutije koja je nelinearna, difuzijski sloj izvodi linearnu operaciju na matricama stanja A i B, odnosno vrijedi $DIFF(A) + DIFF(B) = DIFF(A + B)$.[3]

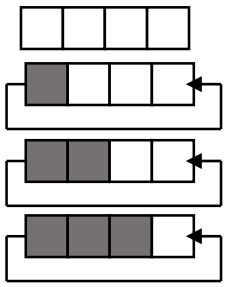
4.2.1. Podsloj ShiftRows

Transformacija ShiftRows ciklički pomiče elemente matrice za određeni broj mjesta ulijevo. Svi unosi koji "otpadnu" ponovo se ubacuju u desnu stranu retka. Pomak se provodi na sljedeći način:

1. prvi red se ne pomiče,
2. drugi red je pomaknut za jednu (bajt) poziciju ulijevo,
3. treći red je pomaknut dvije pozicije ulijevo,
4. četvrti red je pomaknut tri pozicije ulijevo.

Rezultat je nova matrica koja se sastoji od istih 16 bajtova, ali pomaknutih jedan u odnosu na drugi. Na Slici 4.3 prikazana je shema izmjene redova.[4]

B_0	B_4	B_8	B_{12}
B_1	B_5	B_9	B_{13}
B_2	B_6	B_{10}	B_{14}
B_3	B_7	B_{11}	B_{15}



B_0	B_4	B_8	B_{12}
B_5	B_9	B_{13}	B_1
B_{10}	B_{14}	B_2	B_6
B_{15}	B_3	B_7	B_{11}

Slika 4.3 *Shema ShiftRows podsloja*

Primjer 4.2.1.1. Na Primjeru 4.1.3 primijeniti ćemo ShiftRows transformaciju. Nakon zamjene elemenata pomoću S-kutije naša matrica je izgledala ovako:

0A	D4	10	8C
FF	E4	8D	EE
80	81	C6	76
0E	4B	B3	8F

Primjenom ShiftRows transformacije matrica izgleda ovako:

0A	D4	10	8C
E4	8D	EE	FF
C6	76	80	81
8F	0E	4B	B3

4.2.2. Podsloj MixColumn

MixColumn je linearna transformacija koja miješa svaki stupac matrice stanja. MixColumn transformacija predstavlja glavni dio difuzijskog sloja u AES-u budući da svaki ulazni bajt utječe na četiri izlazna bajta. U nastavku ovog teksta ulazno 16-bajtno stanje ćemo označavati sa B (stanje nakon ShiftRows operacije), a izlazno 16-bajtno stanje sa C, odnosno vrijedi $MixColumn(B) = C$.

Svaki stupac matrice se smatra vektorom i množi se s fiksnom matricom veličine $4x4$ koja sadrži konstantne unose. Množenje i zbrajanje koeficijenata vrši se u $GF(2^8)$.

Primjer 4.2.2.1. Prva četiri izlazna bajta računaju se na sljedeći način:

$$\begin{pmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} B_0 \\ B_5 \\ B_{10} \\ B_{15} \end{pmatrix}$$

$$C_0 = 02 \cdot B_0 + 03 \cdot B_5 + 01 \cdot B_{10} + 01 \cdot B_{15}$$

$$C_1 = 01 \cdot B_0 + 02 \cdot B_5 + 03 \cdot B_{10} + 01 \cdot B_{15}$$

$$C_2 = 01 \cdot B_0 + 01 \cdot B_5 + 02 \cdot B_{10} + 03 \cdot B_{15}$$

$$C_3 = 03 \cdot B_0 + 01 \cdot B_5 + 01 \cdot B_{10} + 02 \cdot B_{15}$$

Proračun drugog stupca izlaznih bajtova (C_4, C_5, C_6, C_7) vrši se na način da se konstantna matrica pomnoži sa četiri ulazna bajta (B_4, B_9, B_{14}, B_3). Sa Slike 4.1. možemo vidjeti koji se ulazni bajtovi koriste u svakoj od četiri MixColumn operacije.[4]

Svaki bajt stanja C_i i B_i je 8-bitna vrijednost koja predstavlja element iz $GF(2^8)$. Sva aritmetika koja uključuje koeficijente obavlja se u ovom Galoisovom polju. Što se tiče fiksne

matrice, u njoj su konstantne vrijednosti zapisane u heksadecimalnom zapisu. Na primjer, vrijednost "01" se odnosi na polinom iz $GF(2^8)$ s koeficjentima $(0\ 0\ 0\ 0\ 0\ 0\ 0\ 1)$, odnosno to je element 1 iz Galoisovog polja. Nadalje, vrijednost "02" se odnosi na polinom s koeficijentima $(0\ 0\ 0\ 0\ 0\ 0\ 1\ 0)$, što se odnosi na polinom x, dok se "03" odnosi na element Galoisovog polja $x + 1$.

Zbog lakoće softverske implementacije odabrane su navedene konstante $01, 02$ i 03. Množenje s konstantom vrijednošću 01 je zapravo množenje s identitetom tako da ne zahtjeva nikakvu eksplicitnu operaciju. Množenje s 02 i 03 se može obaviti pomoću tablica s 256 elemenata odnosno dimenzija $16x16$. Također, množenje s 02 se može implementirati kao množenje s polinomom x, te primjenom modularne redukcije s polinomom $P(x) = x^8 + x^4 + x^3 + x + 1$. Isto tako se množenje s 03 obavlja na sličan način. Množenje brojem 03 se može implementirati kao množenje s polinomom $x + 1$ nakon čega slijedi modularna redukcija s $P(x)$. Modularna redukcija je potrebna samo u slučajevima kada imamo polinom čiji je stupanj veći od 7.[3]

Primjer 4.2.2.2. Nastavimo sa Primjerom 4.2.1.1. gdje smo nakon ShiftRows transformacije imali stanje:

0A	D4	10	8C
E4	8D	EE	FF
C6	76	80	81
8F	0E	4B	B3

Slijedi proračun za MixColumn transformaciju:

$$\begin{pmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} 0A \\ E4 \\ C6 \\ 8F \end{pmatrix}$$

$$C_0 = 02 \cdot 0A + 03 \cdot E4 + 01 \cdot C6 + 01 \cdot 8F$$

$$C_1 = 01 \cdot 0A + 02 \cdot E4 + 03 \cdot C6 + 01 \cdot 8F$$

$$C_2 = 01 \cdot 0A + 01 \cdot E4 + 02 \cdot C6 + 03 \cdot 8F$$

$$C_3 = 03 \cdot 0A + 01 \cdot E4 + 01 \cdot C6 + 02 \cdot 8F$$

$\mathbf{02 \cdot 0A} \Rightarrow$ 02 predstavlja polinom x, a vrijednost $0A$ kada prebacimo u binarni sustav dobivamo (0 0 0 0 1 0 1 0) što možemo zapisati u obliku polinoma $(x^3 + x)$ pa slijedi umnožak polinoma $(x^3 + x) \cdot x = x^4 + x^2$

$\mathbf{03 \cdot E4} \Rightarrow$ 03 predstavlja polinom $(x + 1)$, a $E4$ polinom $(x^7 + x^6 + x^5 + x^2)$ pa slijedi umnožak polinoma $(x^7 + x^6 + x^5 + x^2) \cdot (x + 1) = x^8 + x^5 + x^3 + x^2$. Dobiveni polinom je stupnja većeg od 7, te je polinom potrebno modularno reducirati s $P(x)$. Ponekad je jednostavnije reducirati svaki od vodećih članova pojedinačno $\Rightarrow (x^8) \equiv (x^4 + x^3 + x + 1)\ (mod\ P(x))$, dobivamo da je $x^8 + x^5 + x^3 + x^2 = (x^4 + x^3 + x + 1) + x^5 + x^3 + x^2 = x^5 + x^4 + x^2 + x + 1$

$\mathbf{01 \cdot C6} = x^7 + x^6 + x^2 + x$

$\mathbf{01 \cdot 8F} = x^7 + x^3 + x^2 + x + 1$

Dobivamo da je C_0 jednako:

$$\begin{aligned}
\mathbf{02 \cdot 0A} &= & x^4 & & + x^2 & \\
\mathbf{03 \cdot E4} &= & x^5 + x^4 & & + x^2 + x + 1 & \\
\mathbf{01 \cdot C6} &= x^7 + x^6 & & & + x^2 + x & \\
\mathbf{01 \cdot 8F} &= x^7 & & + x^3 + x^2 + x + 1 &
\end{aligned}$$

$$\mathbf{C_0} = x^6 + x^5 + x^3 + x = \mathbf{6A}$$

Proračun ostalih vrijednosti se vrši na isti način, pa se kao rezultat dobiva matrica sa slijedećim vrijednostima:

6A	47	C1	2B
07	41	07	42
F5	A7	38	A4
39	80	C8	8C

4.3. Sloj dodavanja ključa (eng. *Key Addition layer*)

Postoje dva ulaza u sloju za dodavanje ključa:

1. trenutna matrica stanja koja se sastoji od 16 bajtova i
2. potključ koji se također sastoji od 16 bajtova.

U ovom koraku kombiniramo ta dva ulaza pomoću XOR operacije. Važno se je prisjetiti da XOR operacija odgovara zbrajanju u Galoisovom polju $GF(2)$. Proces izračunavanja novog ključa za sljedeće runde poznat je kao raspored ključeva čiji opis slijed u narednom odjeljku. [8]

4.3.1. Raspored ključeva

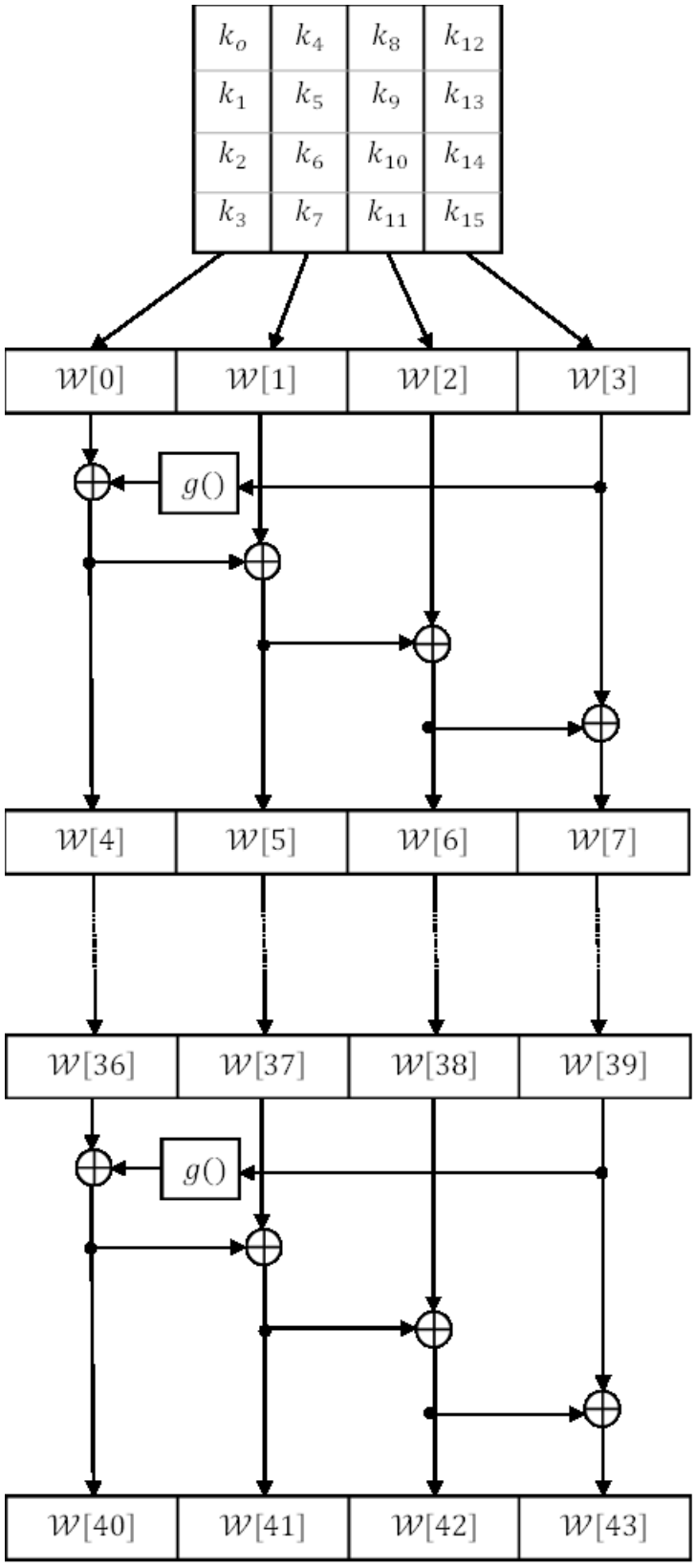
Kod rasporeda ključeva uzima se izvorni ulazni ključ koji može biti duljine 128, 192 ili 256 bita. Iz njega se izvode potključevi koji se koriste u AES-u. XOR operacija za dodavanje potključa se koristi i na ulazu i na izlazu AES-a. Ovaj proces se ponekad naziva "izbjeljivanje" ključeva. Broj potključeva jednak je broju rundi plus jedan. Taj jedan dodatni potključ imamo zbog ključa potrebnog za "izbjeljivanje" u prvom sloju dodavanja ključa (Slika 4.2.).

Kao što možemo vidjeti iz Tablice 2.1 za duljinu ključa od 128 bita broj rundi je $n_r = 10$, a postoji 11 potključeva. Svaki od 11 potključeva se sastoji od 128 bita. AES sa 192-bitnim ključem zahtijeva 13 potključeva duljine 128 bita, a AES s 256-bitnim ključem ima 15 potključeva. Potključevi se izračunavaju rekurzivno, odnosno da bi se izveo potključ k_i, potključ k_{i-1} mora biti poznat itd.[2]

AES raspored ključeva je orijentiran na riječi, gdje je 1 riječ = 32 bita. Potključevi su pohranjeni u polju za proširenje ključeva $\mathcal{W}$ koji se sastoji od riječi. Postoje različiti rasporedi ključeva za tri različite veličine AES ključeva, koji su poprilično slični. U nastavku predstavljamo tri ključna rasporeda.[3]

Raspored ključeva za 128-bitni ključ

Isto kao što je 128-bitni ulazni blok uređen u obliku niza stanja, algoritam raspoređuje prvih 16 bajtova ključa za šifriranje u obliku matrice bajtova veličine $4x4$. Sljedeća Slika 4.4 prikazuje četiri riječi izvornog 128-bitnog ključa koje se proširuju u raspored ključeva koji se sastoji od $4 \cdot 11 = 44$ riječi. Potključevi su pohranjeni u polju za proširenje ključeva, gdje elemente polja označavamo sa $\mathcal{W}[0], \mathcal{W}[1], \dots, \mathcal{W}[43]$. Elementi $k_0, k_1, \dots, k_{15}$ označavaju bajtove izvornog AES ključa. Potključevi se izračunavaju kao što je prikazano na Slici 4.4.[3]



Slika 4.4 *Raspored ključeva za 128-bitnu veličinu ključa*

Sa Slike 4.4 možemo vidjeti da se krajnje lijeve riječi ($\mathcal{W}[4], \ldots, \mathcal{W}[32], \mathcal{W}[36], \mathcal{W}[40]$) označavaju sa $\mathcal{W}[4 \cdot i]$, te izračunavaju na slijedeći način:

$$\mathcal{W}[4 \cdot i] = \mathcal{W}[4 \cdot (i-1)] + g(\mathcal{W}[4 \cdot i - 1]).$$

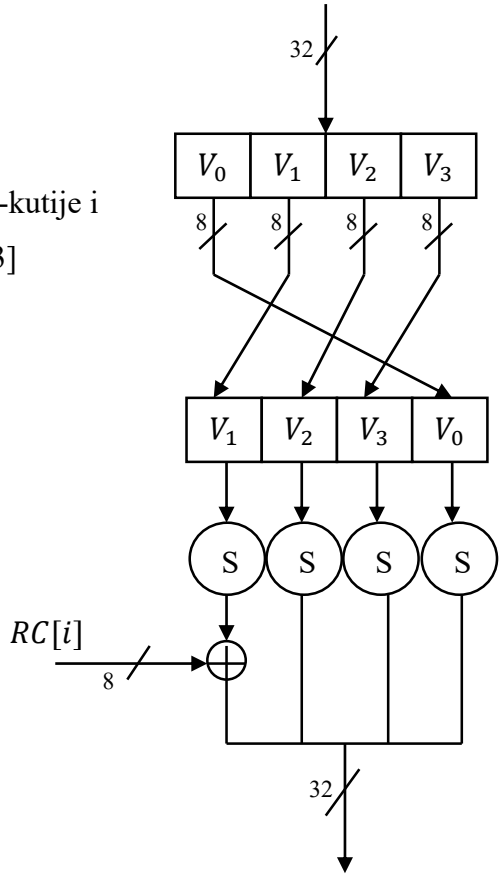
Gdje $g()$ predstavlja nelinearnu funkciju s četverobajtnim ulazom i izlazom. Preostale riječi podključa se izračunavaju rekurzivno kao:

$$\mathcal{W}[4 \cdot i + j] = \mathcal{W}[4 \cdot i + j - 1)] + \mathcal{W}[4 \cdot (i-1) + j],$$

pri čemu je $i = 1,2, \dots, 10$ i $j = 1, 2, 3$.

Funkcija $g()$ sastoji se od tri koraka:

1. najprije rotira svoja četiri ulazna bajta,
2. zatim zamjenjuje svaki bajt u riječi pomoću S-kutije i
3. na kraju na to dodaje okruglu konstantu RC.[3]



Slika 4.5 Shema prikaza funkcije g()

Okrugla konstanta RC je element Galoisovog polja $GF(2^8)$, tj. 8-bitna vrijednost. Predstavlja riječ u kojoj su tri krajnja desna bajta uvijek 0, odnosno dodajemo ju samo krajnjem lijevom bajtu u funkciji $g()$. Okrugla konstanta varira od runde do runde prema sljedećem pravilu:

$$RC[1] = x^0 = (0000\ 0001)_2 = (01)_{hex},$$

$$RC[2] = x^1 = (0000\ 0010)_2 = (02)_{hex},$$

$$RC[3] = x^2 = (0000\ 0100)_2 = (04)_{hex},$$

$$RC[4] = x^3 = (0000\ 1000)_2 = (08)_{hex},$$

$$RC[5] = x^4 = (0001\ 0000)_2 = (10)_{hex},$$

$$RC[6] = x^5 = (0010\ 0000)_2 = (20)_{hex},$$

$$RC[7] = x^6 = (0100\ 0000)_2 = (40)_{hex},$$

$$RC[8] = x^7 = (1000\ 0000)_2 = (80)_{hex},$$

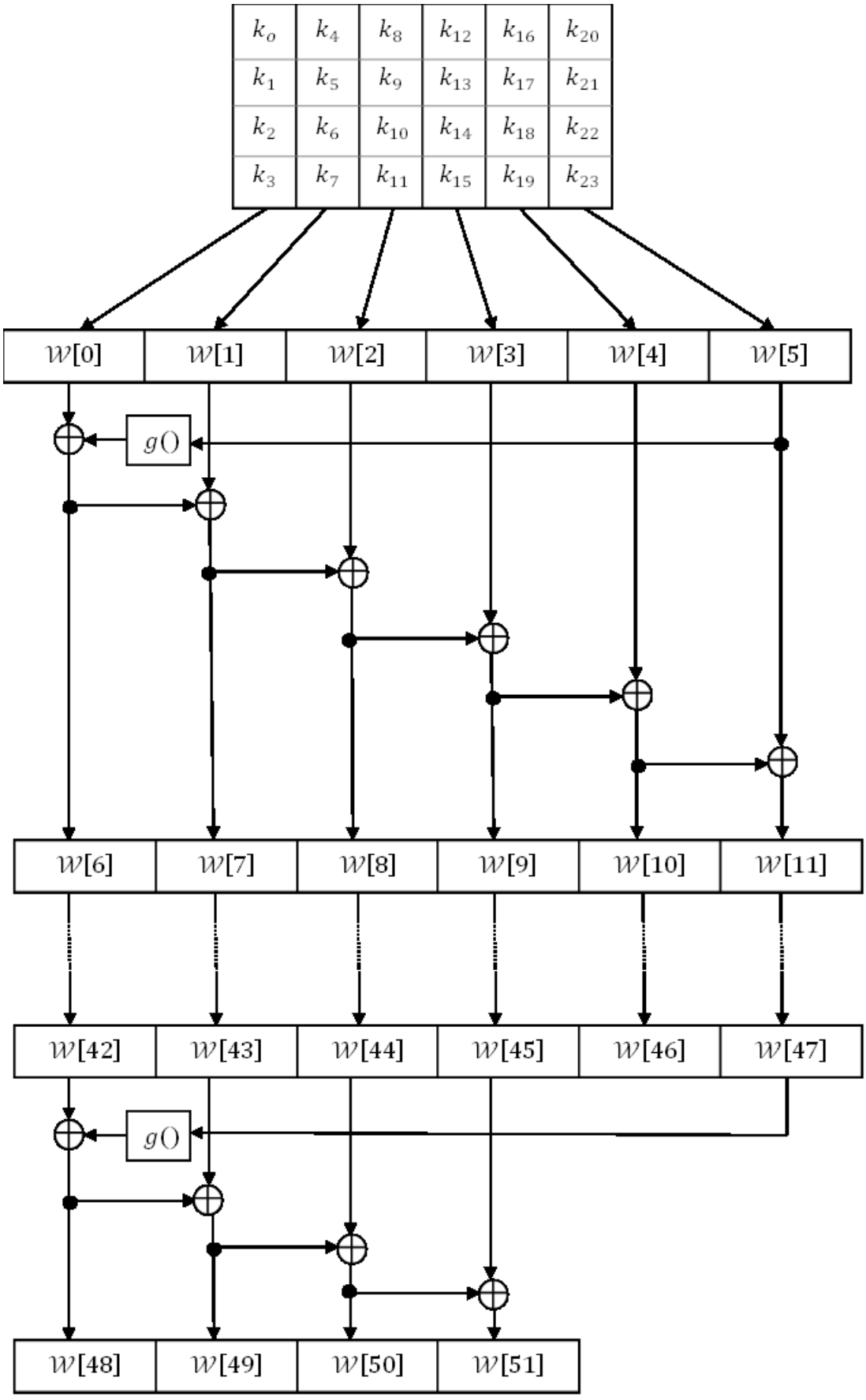
$$RC[9] = x^8 = (0001\ 1011)_2 = (1B)_{hex},$$

$$RC[10] = x^9 = (0011\ 0110)_2 = (36)_{hex}.$$

Funkcija $g()$ nam je vrlo važna jer dodaje nelinearnost u rasporedu ključeva, isto tako uklanja simetriju koja je uvedene u drugim koracima algoritma. Oba svojstva su važna da bi se spriječilo razbijanje šifre, odnosno da bi se spriječili određeni napadi.[3]

Raspored ključeva za 192-bitni ključ

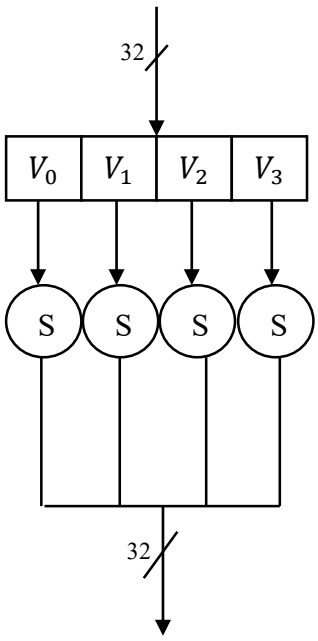
AES sa 192-bitnim ključem ima 12 rundi, tako da postoji 13 potključeva gdje se svaki sastoji od 128 bita. Potključevi zahtijevaju 52 riječi koje su pohranjene u elementima niza $\mathcal{W}[0], \mathcal{W}[1], \dots, \mathcal{W}[51]$. Izračun elemenata niza vrlo je sličan kao kod 128-bitnih ključeva što možemo vidjeti sa Slike 4.6. Za 192-bitni ključ postoji 8 iteracija rasporeda ključeva. Svaka iteracija izračunava šest novih riječi niza potključeva $\mathcal{W}$. . Potključ za prvu rundu AES-a formira se od elemenata niza $(\mathcal{W}[0], \mathcal{W}[1], \mathcal{W}[2], \mathcal{W}[3])$, a drugi potključ se formira od elemenata $(\mathcal{W}[4], \mathcal{W}[5], \mathcal{W}[6], \mathcal{W}[7])$, i tako dalje za ostale potključeve. U funkciji $g()$ potrebno je osam okruglih koeficijenata $RC[i]$. Izračunavaju se kao u slučaju 128-bita i kreću se od $RC[1]$ do $RC[8]$.[3]



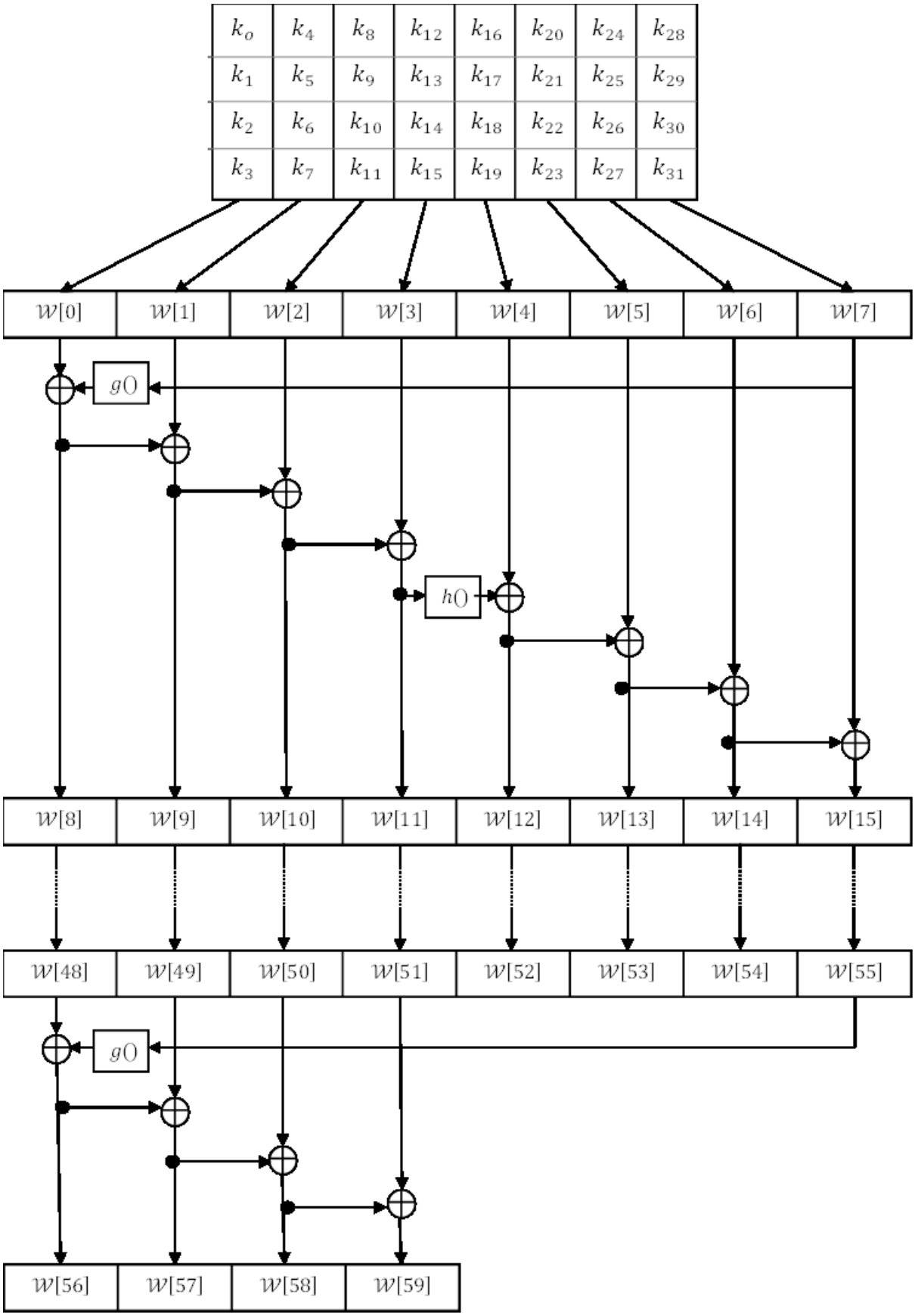
Slika 4.6 *Raspored ključeva za 192-bitnu veličinu ključa*

Raspored ključeva za 256-bitni ključ

AES sa 256-bitnim ključem ima 14 rundi, tako da postoji 15 potključeva gdje se svaki sastoji od 128 bita. Potključevi su pohranjeni u 60 riječi $\mathcal{W}[0], \mathcal{W}[1], \ldots, \mathcal{W}[59]$. Na Slici 4.7 prikazan je proračun elemenata niza koji je vrlo sličan kao u 128-bitnom slučaju. Postoji 7 iteracija za raspored ključeva, gdje svaka iteracija izračunava osam riječi za potključeve. Isto kao i u prethodnom slučaju, potključ za prvu rundu AES-a formira se od elemenata niza $(\mathcal{W}[0], \mathcal{W}[1], \mathcal{W}[2], \mathcal{W}[3])$, a drugi potključ se formira od elemenata $(\mathcal{W}[4], \mathcal{W}[5], \mathcal{W}[6], \mathcal{W}[7])$, i tako dalje za ostale potključeve. U funkciji $g()$ postoji sedam okruglih koeficijenata $RC[1], RC[2] \ldots, RC[7]$ koji se izračunavaju kao u 128-bitnom slučaju. Ovaj raspored ključeva ima i funkciju $h()$ s 4-bajtnim ulazom i izlazom. Funkcija primjenjuje S-Box na sva četiri ulazna bajta.[3]



Slika 4.7 *Shema prikaza funkcije h()*



Slika 4.8 *Raspored ključeva za 256-bitnu veličinu ključa*

Općenito, prilikom implementacije bilo kojeg od rasporeda ključeva, postoje dva različita pristupa:

1. **Predračunanje**

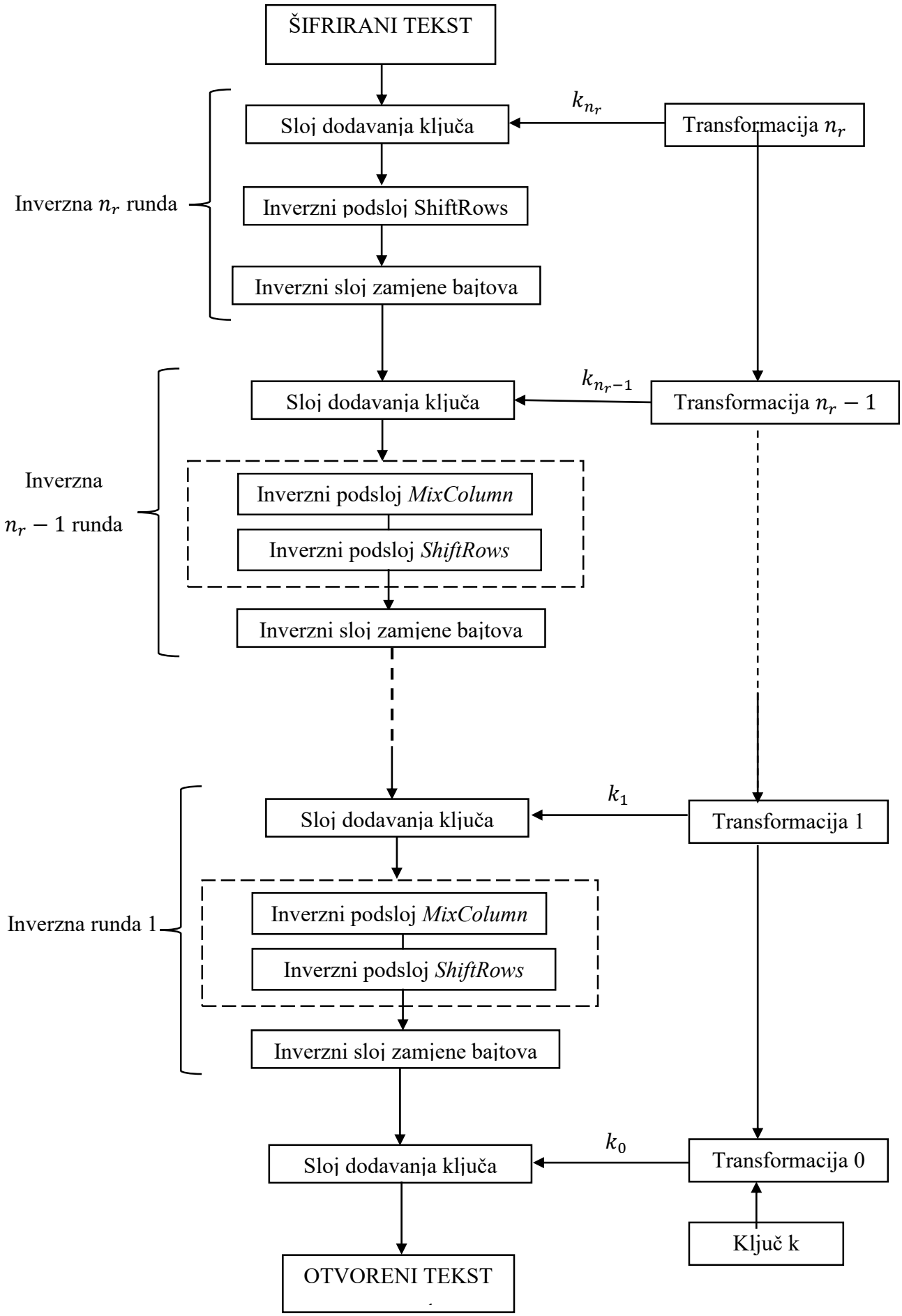
Svi potključevi se najprije proširuju u niz $\mathcal{W}$, dok se šifriranje otvorenog teksta izvršava naknadno. Ovaj pristup se često koristi u računalnim implementacijama AES-a gdje se jednim ključem šifrira velika količina podataka. Važno je uzeti u obzir da ovakav pristup zahtjeva $(n_r + 1) \cdot 16$ bajtova memorije. Odnosno, ukoliko uzmemo 128-bitni ključ tada nam je potrebno $(10 + 1) \cdot 16 = 176$ bajtova. Iz razloga što ćemo često biti ograničeni memorijskim resursima, ovakva implementacija u tim slučajevima nije poželjna.

2. **Računanje u "hodu"**

U svakoj rundi se novi potključ izvodi tijekom šifriranja otvorenog teksta. Moramo uzeti u obzir da se u procesu dešifriranja prvo izvodi XOR operacija između posljednjeg potključa i šifriranog teksta. Iz toga razloga je najprije potrebno rekurzivno izvesti sve potključeve, a zatim započeti s dešifriranjem šifriranog teksta i generiranjem potključeva u "hodu". Ovo rezultira da je proces dešifriranja uvijek nešto sporiji od procesa šifriranja kada se koristi ovaj pristup računanja u "hodu".[3]

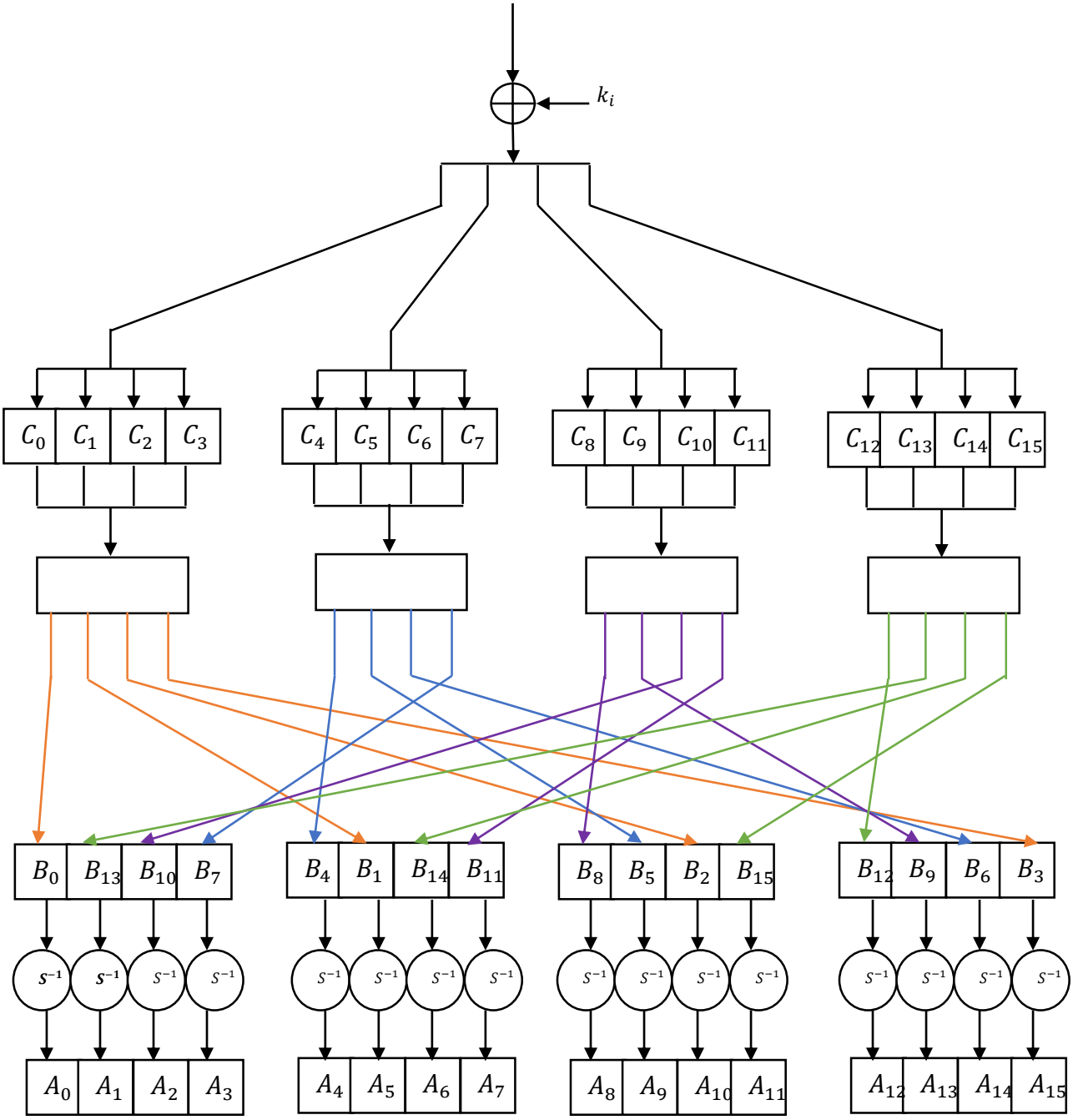
4.4. Dešifriranje

U procesu dešifriranja svi slojevi moraju biti invertirani, što znači da sloj zamjene bajtova postaje Inverzni sloj zamjene bajtova, podsloj ShiftRows postaje Inverzni podsloj ShitRows, a podsloj MixColumn postaje Inverzni podsloj MixColumn. Nadalje ćemo vidjeti da su operacije inverznog sloja prilično slične operacijama koje se koriste prilikom procesa šifriranja. Isto tako važno je istaknuti da je redoslijed ključeva obrnut. Na Slici 4.9 možemo vidjeti blok dijagram funkcije dešifriranja.[2]



Slika 4.9 *Blok dijagram AES dešifriranja*

S obzirom da posljednja runda šifriranja ne izvodi MixColumn operaciju, prva runda dešifriranja također ne sadrži odgovarajući inverzni sloj. Svi ostali krugovi dešifriranja sadrže sve AES slojeve. U nastavku slijedi opis inverznih slojeva općeg AES kruga dešifriranja koji je prikazan Slikom 4.10. Budući da je operacija XOR sama po sebi inverzna, sloj dodavanja ključa će biti isti kod dešifriranja kao što je i kod šifriranja.[2]



Slika 4.10 *Prikaz jedne AES runde dešifriranja*

4.4.1. Inverzni podsloj MixColumn

Nakon dodavanja ključa se primjenjuje inverzni podsloj MixColumn (osim u prvoj rundi deifriranja). U postupku dešifriranja, kako bi se obrnula operacija MixColumn mora se koristiti inverzna vrijednost njegove matrice.

$$\begin{pmatrix} B_0 \\ B_1 \\ B_2 \\ B_3 \end{pmatrix} = \begin{pmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{pmatrix} \begin{pmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \end{pmatrix}$$

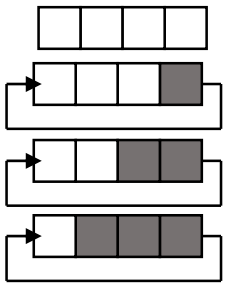
Na ulazu imamo 4-bajtni stupac stanja C koji se množi s inverznom matricom $4x4$. Matrica dimenzije $4x4$ sadrži konstantne unose, a operacije zbrajanja i množenja koeficijenata se vrši u $GF(2^8)$.

Drugi stupac izlaznih bajtova B_4, B_5, B_6, B_7 se izračunava množenjem slijedeća četiri ulazna bajta C_4, C, C_6, C_7 sa istom konstantnom matricom i tako dalje. Svaka vrijednost B_i i C_i kao i konstante su elementi iz $GF(2^8)$. Konstante su zapisane u heksadecimalnom brojevnom sustavu.[4]

4.4.2. Inverzni podsloj ShiftRows

Kod inverznog ShiftRows podsloja retke matrice stanja pomičemo u suprotnom smjeru. Prvi redak ostaje isti, drugi se pomiče za jedno mjesto u desno, treći za dva mjesta, a četvrti za tri mjesta u desno. Shema pomicanja elemenata u redcima prikaza na je Slikom 4.11.[4]

B_0	B_4	B_8	B_{12}
B_1	B_5	B_9	B_{13}
B_2	B_6	B_{10}	B_{14}
B_3	B_7	B_{11}	B_{15}



B_0	B_4	B_8	B_{12}
B_{13}	B_1	B_5	B_9
B_{10}	B_{14}	B_2	B_6
B_7	B_{11}	B_{15}	B_3

Slika 4.11 *Shema Inverznog ShiftRows podsloja*

4.4.3. Inverzni sloj zamjene bajtova

U ovom koraku se prilikom dešifriranja koristi inverz od S-kutije.

Tablica 4.4 *Prikaz inverznog S-Box-a*

X	Y															
	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	FA	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

4.4.4. Raspored ključeva za dešifriranje

S obzirom da u prvom krugu dešifriranja koristimo zadnji potključ, u drugom krugu dešifriranja pretposljednji ključ i tako dalje, možemo zaključiti da nam je potreban potključ obrnutim redoslijedom kao što je prikazano na Slici 4.9. To se postiže tako što se uglavnom prvo izračuna cijeli raspored ključeva i pohrani svih 11, 13 ili 15 potključeva, ovisno o broju krugova koje AES koristi odnosno o duljini ključeva koje podržava AES. Ovo preračunavanje dodaje vremensko čekanje zbog kojeg je postupak dešifriranja sporiji od postupka šifriranja.[3]

5. NAPADI NA AES

AES još uvijek nije probijen na isti način na koji je DES bio 1999. godine, a najveći uspješni napad grubom silom na bilo koju blok je šifru bio samo protiv 64-bitne enkripcije. Napad grubom silom je metoda hakiranja koja koristi pokušaje i pogreške za probijanje lozinki, vjerodajnica za prijavu i ključeva za šifriranje. To je jednostavna, ali pouzdana taktika za dobivanje neovlaštenog pristupa pojedinačnim računima i sustavima te mrežama organizacija. Haker pokušava s više korisničkih imena i lozinki, često koristeći računalo za testiranje širokog raspona kombinacija, sve dok ne pronađe točne podatke za prijavu.

Istraživanje napada na AES enkripciju nastavljeno je otkako je standard finaliziran 2000. godine. Razni istraživači objavili su napade na smanjene verzije AES-a. Pronađeno je nekoliko potencijalnih načina za napad na AES enkripciju:

- 2009. otkrili su mogući napad s povezanim ključem. Ova kriptoanaliza pokušala je razbiti šifru proučavajući kako ona djeluje koristeći različite ključeve. Napad povezanim ključem pokazao se prijetnjom samo za AES sustave koji su pogrešno konfigurirani.
- Godine 2009. došlo je do napada poznatog ključa na AES-128. Poznati ključ korišten je za razlučivanje strukture enkripcije. Međutim, haker je ciljao samo na verziju AES-128 s osam rundi, umjesto na standardnu verziju od 10 rundi što je prijetnju činilo relativno malom.[11]

Većina kriptografa se slaže da bi uz trenutni hardver uspješan napad na AES algoritam, čak i na 128-bitnom ključu, trajao milijarde godina i stoga je vrlo nevjerojatan. U ovom trenutku ne postoji niti jedna poznata metoda koja bi nekome omogućila napad i dešifriranje podataka šifriranih AES-om sve dok je algoritam pravilno implementiran.

Unatoč svim dokazima koji ukazuju na nepraktičnost AES napada s trenutnim hardverom, to ne znači da je AES potpuno siguran. Napadi sa strane kanala, koji se temelje na informacijama dobivenim iz fizičke implementacije kriptosustava se još uvijek mogu iskoristiti za napad na sustav šifriran s AES-om. Ti se napadi ne temelje na slabostima algoritma, već na fizičkim pokazateljima potencijalne slabosti koja se može iskoristiti za probijanje sustava.

Slijedi nekoliko uobičajenih primjera napada sa strane kanala:

- Napad na vrijeme- ovi napadi se temelje na tome da napadači mjere koliko je potrebno vremena za izvođenje različitih izračuna;

- Napad praćenja napajanja- ovi se napadi oslanjaju na varijabilnosti potrošnje električne energije tijekom izračuna;
- Elektromagnetski napadi- ovi napadi se temelje na mjerenju elektromagnetskog zračenja opreme.

Stručnjaci za sigurnost smatraju da je AES siguran kada se pravilno implementira. Međutim, ključevi za AES šifriranje moraju biti zaštićeni. Čak i najopsežniji kriptografski sustavi mogu biti ranjivi ako haker dobije pristup ključu za šifriranje. Kako bismo osigurali sigurnost AES ključeva možemo utjecati na način da koristimo jake lozinke, koristimo upravitelje lozinki, zahtijevamo višefaktorsku provjeru autentičnosti te postavljanjem vatrozida i antivirusnih programa. Važno je provođenje obuke za podizanje savjesti o sigurnosti kako bi se spriječilo da pojedinci postanu žrtve socijalnog inženjeringa i krađe identiteta.[11]

6. PROGRAMSKO RJEŠENJE

6.1. Java programski jezik

Programski jezik korišten u realizaciji ovoga programskog rješenja jest programski jezik Java. To je programski jezik opće namjene, brz, siguran, pouzdan, neovisan o platformi, te objektno orijentiran. Java je objektno orijentiran programski jezik visoke razine koji se temelji na klasi i koji je dizajniran da ima što je moguće manje ovisnosti o implementaciji. To je programski jezik opće namjene namijenjen da programerima omogući pisanje jednom, pokretanje bilo gdje (WORA), što znači da se prevedeni Java kod može izvoditi na svim platformama koje podržavaju Javu bez potrebe za ponovnim kompajliranjem.

6.2. Opis rješenja

Za šifriranje u AES-u ovo rješenje koristi 4 klase:

- *AES_sifriranje_desifriranje* klasa pruža sve glavne funkcije potrebne za AES algoritam šifriranja i dešifriranja.
- *Tablice* klasa daje pristup izračunatim tablicama potrebnim za izvođenje AES funkcija.
- *Skeniranje* klasa kopira i ispisuje nizova bajtova.
- *Test* klasa predstavlja pokretački program za testiranje procesa šifriranja i dešifriranja.

Predloženi sustav radi sa 128-bitnom veličinom bloka kao i sa 128-bitnom veličinom ključa. Algoritam se primjenjuje i za šifriranje i za dešifriranje teksta. Kako je veličina ključa 128 bita, bit će potrebno 10 rundi.

6.2.1. AES_sifriranje_desifriranje.java

U klasi *AES_sifriranje_desifriranje* koja je prikazana na Slici 6.1 vidimo konstruktor koji služi za inicijalizaciju atributa. Inicijalizirali smo broj riječi u ključu, broj rundi koji ovisi o duljini ključa (u našem slučaju uvijek će biti 10 rundi), prostor za niz *w* koji predstavlja niz ključeva, te funkciju *KeyExpansion*.

```java
package Package;

public class AES_sifriranje_desifriranje {
    private final int Nb = 4;   // Broj stupaca (32-bitne riječi) koji čine stanje, Nb = 4.
    private int Nk;             // Broj 32-bitnih riječi koje čine ključ šifre, Nk = 4, 6, ili 8.
    private int Nr;             // Broj rundi, Nr = 10, 12, ili 14.
    private int wBr;            // pozicija u w za RoundKey
    private Tablice T;          // Tlice potrebne za AES
    private byte[] w;           // proširenI ključ (the expanded key)

    // konstruktor za klasu.
    public AES_sifriranje_desifriranje(byte[] kljuc, int NkIn) {
        Nk = NkIn;                  // Broj riječi u ključu 4, 6 ili 8
        Nr = Nk + 6;                // broj rundi
        T = new Tablice();          // klasa za dobivanje vrijednosti iz Tlica
        w = new byte[4*Nb*(Nr+1)];  // niz za prošireni ključ
        KeyExpansion(kljuc, w);     // duljina od w ovisi o broju rundi

```

Slika 6.1 *Konstruktor za klasu AES_sifriranje_desifriranje*

Naredna Slika 6.2 prikazuje funkciju AES šifriranja. Najprije se vrši kopiranje ulaznog teksta u matricu stanja (pomoću klase skeniranje u kojoj se nalazi metoda kopija) i *AddRoundKey* transformacije. Zatim se devet puta izvršavaju AES slojevi (*SubBytes, ShiftRowy, MixColumns* i *AddRoundKey*). U desetoj rundi izostavljena je *MixColumns* transformacija baš kao što je prikazano blok shemom na Slici 2.2.

```java
    // AES šifriranje, metoda koja se koristi za šifriranje otvorenog teksta pomoću danog ključa.
    public void Sifriranje(byte[] ulaz, byte[] izlaz) {
        wBr = 0;                        // broji bajtove u proširenom ključu trijekom širiranja
        byte[][] stanje = new byte[4][Nb];   // matrica stanja
        skeniranje.kopija(stanje, ulaz);     // kopija komponenti iu ulaza u stanje
        AddRoundKey(stanje);                 // xor sa proširenim ključem
        //Nr runde širiranja
        for (int runda = 1; runda < Nr; runda++) {
            System.out.println(x: "--------------------------------------------------------------------------------");
            skeniranje.ispisNiza("Pocetak " + runda + ". runde:    ", stanje);
            SubBytes(stanje); // sloj zamjene bajtova, S-box
            skeniranje.ispisNiza(naziv: "SubBytes:            ", stanje);
            ShiftRows(stanje); // ShiftRows transormacija
            skeniranje.ispisNiza(naziv: "ShitRow:             ", stanje);
            MixColumns(stanje); // MixColumn transformacija
            skeniranje.ispisNiza(naziv: "MixColumn:           ", stanje);
            AddRoundKey(stanje); // xor sa proširenim ključem

        }
        //Zadnja runda, nema MixColumn transformaciju
        System.out.println(x: "--------------------------------------------------------------------------------");
        skeniranje.ispisNiza("Pocetak " + Nr + ". runde:   ", stanje);
        SubBytes(stanje); // sloj zamjene bajtova, S-box
        skeniranje.ispisNiza(naziv: "SubBytes:            ", stanje);
        ShiftRows(stanje); // ShiftRows transormacija
        skeniranje.ispisNiza(naziv: "ShiftRow:            ", stanje);
        AddRoundKey(stanje); //xor sa proširenim ključem
        skeniranje.kopija(izlaz, stanje);
    }
```

Slika 6.2 *Metoda Sifriranje*

Na Slici 6.3 prikazana je metoda AES dešifriranja, koja je obrnuta od metode šifriranja. U prvoj rundi se ne izvršava MixColumn transformacija, dok se u ostalima izvršava.

```
51      // AES dešifriranje, metoda koja se koristi za dešifriranje šifriranog teksta pomoću danog ključa.
52      public void deSifriranje(byte[] ulaz, byte[] izlaz) {
53          wBr = 4*Nb*(Nr+1);                  // brojanje bajtova u proširenom ključu tijekom širiranja
54          byte[][] stanje = new byte[4][Nb];   // matrica stanja
55          skeniranje.kopija(stanje, ulaz);     // kopija komponenti
56          InvAddRoundKey(stanje);              // xor sa proširenim ključem
57
58          for (int runda = Nr-1; runda >= 1; runda--) {
59              System.out.println(x: "------------------------------------------------------------------------");
60
61              skeniranje.ispisNiza("Pocetak " + (Nr - runda) + ". runde:    ", stanje);
62              InvShiftRows(stanje); // inverzna ShiftRows transormacija
63              skeniranje.ispisNiza(naziv: "invSbox:             ", stanje);
64              InvSubBytes(stanje); // inverzni sloj zamjene bajtova, S-box
65              skeniranje.ispisNiza(naziv: "invSubBytes:         ", stanje);
66              InvAddRoundKey(stanje); // xor sa proširenim ključem
67              InvMixColumns(stanje); // inverzna MixColumn transformacija
68              skeniranje.ispisNiza(naziv: "invMixColumn:        ", stanje);
69          }
70          System.out.println(x: "------------------------------------------------------------------------");
71          skeniranje.ispisNiza("Pocetak " + Nr + ". runde:    ", stanje);
72          InvShiftRows(stanje); // inverzna ShiftRows transormacija
73          skeniranje.ispisNiza(naziv: "invSbox:             ", stanje);
74          InvSubBytes(stanje); // inverzni sloj zamjene bajtova, S-box
75          skeniranje.ispisNiza(naziv: "invSubBytes:         ", stanje);
76          InvAddRoundKey(stanje); // xor sa proširenim ključem
77          skeniranje.kopija(izlaz, stanje);
78      }
```

Slika 6.3 *Metoda deSifriranje*

U slučaju AES-a, postoji nekoliko krugova, od kojih svaki treba svoj ključ, tako da je stvarni ključ "razvučen" i transformiran kako bi se dobili dijelovi ključa za svaki krug. Najprije se kopira ulazni ključ u niz *w*, te se pomoćni niz *pr* koji se sastoji od 4 bajta puni sa zadnjom riječi iz niza *w*. Pomoćna varijabla *i* nakon svake treće riječi nad nizom *pr* vrši xor između izrotiranih bajtova za 1 mjesto u lijevo i primjene transformacije zamjene bajtova pomoću Sbox-a sa, te niza *RC* s druge strane. Na kraju se svaka riječ dobiva kao rezultat xor operacije prethodne riječi sa pomoćnim nizom *pr*. Sve je to vidljivo u kodu na Slici 6.4.

```
80      // KeyExpansion: proširenje ključa, bajt orijentirani kod, ali prati riječi
81      private void KeyExpansion(byte[] kljuc, byte[] w) {
82          byte[] pr = new byte[4];
83          // prvo kopirati ključ u niz w
84          int j = 0;
85          while (j < 4*Nk) {
86              w[j] = kljuc[j++];
87          }
88          int i;
89          while(j < 4*Nb*(Nr+1)) {
90              i = j/4; // j je ovdje uvijek višekratnik od 4
91              // obraditi svaku riječ, po 4 bajta odjednom
92              for (int prX = 0; prX < 4; prX++)
93                  pr[prX] = w[j-4+prX];
94              if (i % Nk == 0) {
95                  byte tmp, prRC;
96                  byte pr0 = pr[0];
97                  for (int prX = 0; prX < 4; prX++) {
98                      if (prX == 3) tmp = pr0;
99                      else tmp = pr[prX+1];
100                     if (prX == 0) prRC = T.RC(i/Nk);
101                     else prRC = 0;
102                     pr[prX] = (byte)(T.SBox(tmp) ^ prRC);
103                 }
104             }
105             else if (Nk > 6 && (i%Nk) == 4) {
106                 for (int prX = 0; prX < 4; prX++)
107                     pr[prX] = T.SBox(pr[prX]);
108             }
109             for (int prX = 0; prX < 4; prX++)
110                 w[j+prX] = (byte)(w[j - 4*Nk + prX] ^ pr[prX]);
111
112             j = j + 4;
113
114         }
115     }
116
```

Slika 6.4 *Metoda KeyExpansion*

Metode *SubBytes* i *InvSubBytes* prikazane Slikom 6.5 uključuju nelinearnu zamjenu bajtova, djelujući na svakom od bajtova stanja neovisno. Vrši se korištenjem izračunate tablice zamjene bajtova koja se nalazi u klasi *Tablice*. *Sbox* tablica sadrži 256 brojeva (od 0 do 255) i njihove odgovarajuće rezultirajuće vrijednosti.

```
116
117     // SubBytes: primijena Sbox zamjenu na svaki bajt stanja
118     private void SubBytes(byte[][] stanje) {
119         for (int red = 0; red < 4; red++)
120             for (int stupac = 0; stupac < Nb; stupac++)
121                 stanje[red][stupac] = T.SBox(stanje[red][stupac]);
122     }
123
124     // InvSubBytes: primjena inverzne Sbox zamjene na svaki bajt stanja
125     private void InvSubBytes(byte[][] stanje) {
126         for (int red = 0; red < 4; red++)
127             for (int stupac = 0; stupac < Nb; stupac++)
128                 stanje[red][stupac] = T.invSBox(stanje[red][stupac]);
129     }
130
```

Slika 6.5 *Metode SubBytes i InvSubBytes*

ShitRows i *InvShiftRows* metode prikazane na Slici 6.6 pomiču retke matrice kako je to objašnjeno u odjeljku 4.2.1 i 4.4.2 ovoga rada.

```
131     // ShiftRows: zadnja tri retka stanja se pomču za određeni broj koraka
132     private void ShiftRows(byte[][] stanje) {
133         byte[] tem = new byte[4];
134             for (int r = 1; r < 4; r++) {
135                 for (int s = 0; s < Nb; s++)
136                     tem[s] = stanje[r][(s + r)%Nb];
137                 for (int s = 0; s < Nb; s++)
138                     stanje[r][s] = tem[s];
139             }
140     }
141 
142     // InvShiftRows: pomiče retke stanja za određeni broj koraka
143     private void InvShiftRows(byte[][] stanje) {
144         byte[] tem = new byte[4];
145         for (int r = 1; r < 4; r++) {
146             for (int s = 0; s < Nb; s++)
147                 tem[(s + r)%Nb] = stanje[r][s];
148             for (int s = 0; s < Nb; s++)
149                 stanje[r][s] = tem[s];
150         }
151     }
152 
```

Slika 6.6 *Metode ShiftRows i InvShiftRows*

Slika 6.7 prikazuje metodu *MixColumns* koja vrši promjene nad stupcima matrice stanja, a cijela pozadina objašnjena je u odjeljku 4.2.2.

```
153     // MixColumns transormacija
154     private void MixColumns(byte[][] c) {
155         int[] s = new int[4];
156         for (int col = 0; col < 4; col++) {
157             s[0] = T.Mul((byte)0x02, c[0][col]) ^ T.Mul((byte)0x03, c[1][col]) ^ T.Mul((byte)0x01, c[2][col]) ^ T.Mul((byte)0x01, c[3][col]);
158             s[1] = T.Mul((byte)0x01, c[0][col]) ^ T.Mul((byte)0x02, c[1][col]) ^ T.Mul((byte)0x03, c[2][col]) ^ T.Mul((byte)0x01, c[3][col]);
159             s[2] = T.Mul((byte)0x01, c[0][col]) ^ T.Mul((byte)0x01, c[1][col]) ^ T.Mul((byte)0x02, c[2][col]) ^ T.Mul((byte)0x03, c[3][col]);
160             s[3] = T.Mul((byte)0x03, c[0][col]) ^ T.Mul((byte)0x01, c[1][col]) ^ T.Mul((byte)0x01, c[2][col]) ^ T.Mul((byte)0x02, c[3][col]);
161             for (int i = 0; i < 4; i++) c[i][col] = (byte)(s[i]);
162         }
163     }
164 
165     // InvMixColumns transformacija
166     private void InvMixColumns(byte[][] c) {
167         int[] s = new int[4];
168         for (int col = 0; col < 4; col++) {
169             s[0] = T.Mul((byte)0x0e, c[0][col]) ^ T.Mul((byte)0x0b, c[1][col]) ^ T.Mul((byte)0x0d, c[2][col]) ^ T.Mul((byte)0x09, c[3][col]);
170             s[1] = T.Mul((byte)0x09, c[0][col]) ^ T.Mul((byte)0x0e, c[1][col]) ^ T.Mul((byte)0x0b, c[2][col]) ^ T.Mul((byte)0x0d, c[3][col]);
171             s[2] = T.Mul((byte)0x0d, c[0][col]) ^ T.Mul((byte)0x09, c[1][col]) ^ T.Mul((byte)0x0e, c[2][col]) ^ T.Mul((byte)0x0b, c[3][col]);
172             s[3] = T.Mul((byte)0x0b, c[0][col]) ^ T.Mul((byte)0x0d, c[1][col]) ^ T.Mul((byte)0x09, c[2][col]) ^ T.Mul((byte)0x0e, c[3][col]);
173             for (int i = 0; i < 4; i++) c[i][col] = (byte)(s[i]);
174         }
175     }
176 
```

Slika 6.7 *Metode MixColumns i InvMixColumns*

Metoda *AddRoundKey* vrši xor operaciju između trenutne matice stanja i okruglog ključa.

```
177     // AddRoundKey: stanje se kombinira s blokom okruglog ključa koristeći  xor
178     private void AddRoundKey(byte[][] stanje) {
179         for (int s = 0; s < Nb; s++)
180             for (int r = 0; r < 4; r++)
181                 stanje[r][s] = (byte)(stanje[r][s] ^ w[wBr++]);
182     }
183     // InvAddRoundKey: isto kao AddRoundKey, ali unatrag
184     private void InvAddRoundKey(byte[][] stanje) {
185         for (int s = Nb - 1; s >= 0; s--)
186             for (int r = 3; r >= 0 ; r--)
187                 stanje[r][s] = (byte)(stanje[r][s] ^ w[--wBr]);
188     }
```

Slika 6.8 *Metode AddRoundKey i InvAddRoundKey*

6.2.2. Tablice.java

U klasi *Tablice* nalazi se konstruktor u kojem se nalazi pet metoda čija je uloga popuniti niz koji predstavlja određenu tablicu kako bi imali mogunost čitanja podataka iz tih tablica.

```
1   package Package;
2
3   public class Tablice {
4
5       public Tablice() {
6           Etablica();
7           Ltablica();
8           Stablica();
9           invStablica();
10          Powtablica();
11      }
12          private byte[] EXP = new byte[256];      // exp tablica
13          private byte[] LOG = new byte[256];      // Log tablica
14          private byte[] S = new byte[256];        // Sbox tablica
15          private byte[] invS = new byte[256];     // inverzna Sbox tablica
16          private byte[] powX = new byte[15];      // potencija od x = 0x02
17
18          // Metode za pristup sadržaju u tablici
19          public byte SBox(byte b) {
20              return S[b & 0xff];
21          }
22          public byte invSBox(byte b) {
23              return invS[b & 0xff];
24          }
25          public byte RC(int i) {
26              return powX[i-1];
27          }
28
```

Slika 6.9 *Konstruktor klase Tablice i metode za pristup tablicama*

Metoda *Mul* je funkciju koja izvodi množenje u polju $GF(2^8)$ što je omogućenom uporabom pomaka bitova. Pomak se vrši nad polinomom *a* za jedno mjesto u desno. Ukoliko se uslijed pomicanja dogodi da je zadnji bit od polinoma *a* jednak 1, vrši se xor operacija ostatka sa polinomom *b*. U kodu na slici 6.10 je vidljivo da se stanje od krajnjeg desnog bita varijable *b* pohranjuje u privremenoj varijabli *t*, nakon čega polinom *b* pomičemo za 1 bit u lijevo. Može se dogoditi da prilikom pomicanja polinom *b* postane polinom 8. stupnja, tada je potrebno izvršiti xor operaciju između *b* i ireducibilnog polinoma.

```
30      //Funkcija množenja dva polinoma preko GF(2^8) koja se koristi u MixColumns
31      public byte Mul(byte a, byte b) {
32      byte r = 0, t;
33
34          while (a != 0) {
35              if ((a & 1) != 0)
36                  r = (byte) (r ^ b);
37              t = (byte) (b & 0x80);
38              b = (byte) (b << 1);
39              //Ako je rezultat 8. stupnja dodaj P(x)
40              if (t != 0)
41                  b = (byte) (b ^ 0x1b);
42              a = (byte) ((a & 0xff) >> 1);
43          }
44
45      return r;
46      }
47
```

Slika 6.10 *Metoda Mul*

U *Etablice* vrši se potenciranje polinoma 0x03 sa 256 vrijednosti.

```
48      //kreira i popunjava teblicu EXP (tablica potencija)
49      private void Etablica() {
50          int i = 0;
51          byte x = (byte)0x01;
52          EXP[i++] = (byte)0x01;
53          for (int j = 0; j < 255; j++) {
54              byte y = Mul(x, (byte)0x03);
55              EXP[i++] = y;
56              x = y;
57          }
58      }
59
```

Slika 6.11 *Metoda Etablice*

Iz *Etablice* izvedena je logaritamska tablica *Ltablica.*

```
59
60          //pomoću EXP tablice popunjavamo LOG tablicu
61          private void Ltablica() {
62              for (int i = 0; i < 255; i++) {
63                  LOG[EXP[i] & 0xff] = (byte)i;
64              }
65          }
66
```

Slika 6.12 *Metoda Ltablice*

Metoda *SBox* jednostavno uzima odgovarajuću vrijednost iz niza *S*. Sadrži rezultate operacije *subBytes* primijenjenih na svakom bajtu. Metoda *invSBox* uzima odgovarajuću vrijednost iz *invS* niza u kojem se nalaze inverzne vrijednosti niza *S*. Metode *Stablica* i *invStablica* služe za popunjavanje vrijednosti nizova.

```
67      // popunjavanje S tablicu
68      private void Stablica() {
69          for (int i = 0; i < 256; i++)
70              S[i] = (byte)(subBytes((byte)(i & 0xff)) & 0xff);
71      }
72
73
74      //punjenje inverzne S tablice pomoću S tablice
75      private void invStablica() {
76          for (int i = 0; i < 256; i++) {
77              invS[S[i] & 0xff] = (byte)i;
78          }
79      }
80
```

Slika 6.13 *Metode Stablica i invStablica*

U *powX* nizu pohranjene su vrijednosti potencije polinoma 0x02 od 0x0000 do 0x00ff.

```
81      //punjenje powX tablice množenjem polinoma
82       private void Powtablica() {
83           byte x = (byte)0x02;
84           byte xp = x;
85           powX[0] = 1;
86           powX[1] = x;
87           for (int i = 2; i < 15; i++) {
88               xp = Mul(xp, x);
89               powX[i] = xp;
90           }
91       }
92
```

Slika 6.14 *Metoda Powtablica*

Metoda *Inv* predstavlja multiplikativni inverz za bajt koji nam je potreban za izvođenje *subBytes* funkcije.

```
93          // multiplikativni inverz za bajt
94          public byte Inv(byte b) {
95              byte e = LOG[b & 0xff];
96              return EXP[0xff - (e & 0xff)];
97          }
98
```

Slika 6.15 *Metoda Inv*

SubBytes funkcija služi za izračun vrijednosti potrebnih zaa popunjavanje Sbox tablice po pravilu kako je objašnjeno u dijelu 4.1. ovoga rada.

```java
// subBytes funkcija
public int subBytes(byte b) {
    int rez = 0;
    if (b != 0)
        b = (byte)(Inv(b) & 0xff);
    byte c = (byte)0x63;
    for (int i = 0; i < 8; i++) {
        int tmp = 0;
        tmp = iBit(b, i) ^ iBit(b, (i+4)%8) ^ iBit(b, (i+5)%8) ^ iBit(b, (i+6)%8) ^ iBit(b, (i+7)%8) ^ iBit(c, i);
        rez = rez | (tmp << i);
    }
    return rez;
}

    // vraća i-ti bit iz bajta
    public int iBit(byte b, int i) {
        int m[] = {0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80};
        return (b & m[i]) >> i;
    }
```

Slika 6.16 *subBytes*

6.2.3 skeniranje.java

U klasi *skeniranje* nalaze se dvije metode *kopija* koje služe za kopiranje ulaznih podataka u matricu stanja i kopiranju izlaznih podataka iz matrice stanja. Također se nalaze dvije metode *ispisNiza* koje služe za ispis bajtova u heksadecimalnom obliku. Metoda *hex* uzima četiri bita od ulaznog bajta i čita prvu heksadecimalnu znamenku iz niza *dig*, na isti način za četiri desna preostala bita.

```java
package Package;

public class skeniranje {
    private static final int Nb = 4;
    private static String[] dig = {"0","1","2","3","4","5","6","7","8","9","a","b","c","d","e","f"};
    // hex: ispisuje bajt kao dvije heksadecimalne znamenke
    public static String hex(byte a) {
        return dig[(a & 0xff) >> 4] + dig[a & 0x0f];
    }
    //ispisati nizove bajtova
    public static void ispisNiza(String naziv, byte[] a) {
        System.out.print(naziv + " ");
        for (int i = 0; i < a.length; i++)
            System.out.print(hex(a[i]) + " ");
        System.out.println();
    }
    public static void ispisNiza(String naziv, byte[][] s) {
        System.out.print(naziv + " ");
        for (int c = 0; c < Nb; c++)
            for (int r = 0; r < 4; r++)
                System.out.print(hex(s[r][c]) + " ");
            System.out.println();
    }
    // kopirati u stanje
    public static void kopija(byte[][] stanje, byte[] ulaz) {
        int ulazLok = 0;
        for (int c = 0; c < Nb; c++)
            for (int r = 0; r < 4; r++)
                stanje[r][c] = ulaz[ulazLok++];
    }
    // kopirati stanje u izlaz
    public static void kopija(byte[] izlaz, byte[][] stanje) {
        int izlazLok = 0;
        for (int c = 0; c < Nb; c++)
            for (int r = 0; r < 4; r++)
                izlaz[izlazLok++] = stanje[r][c];
    }
}

```

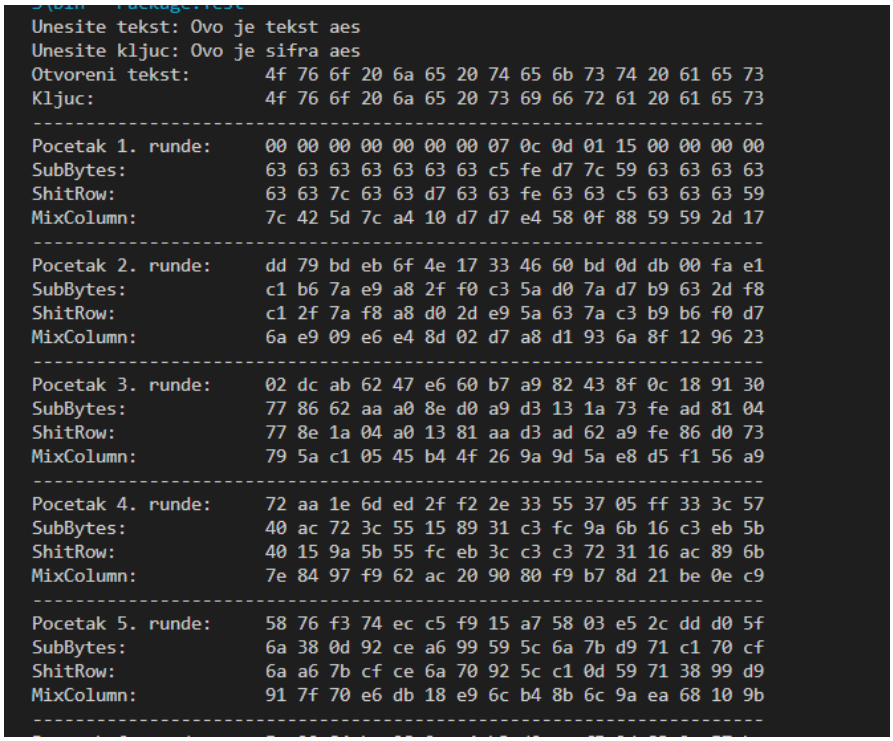
Slika 6.17 *Klasa skeniranje*

6.2.4. Test.java

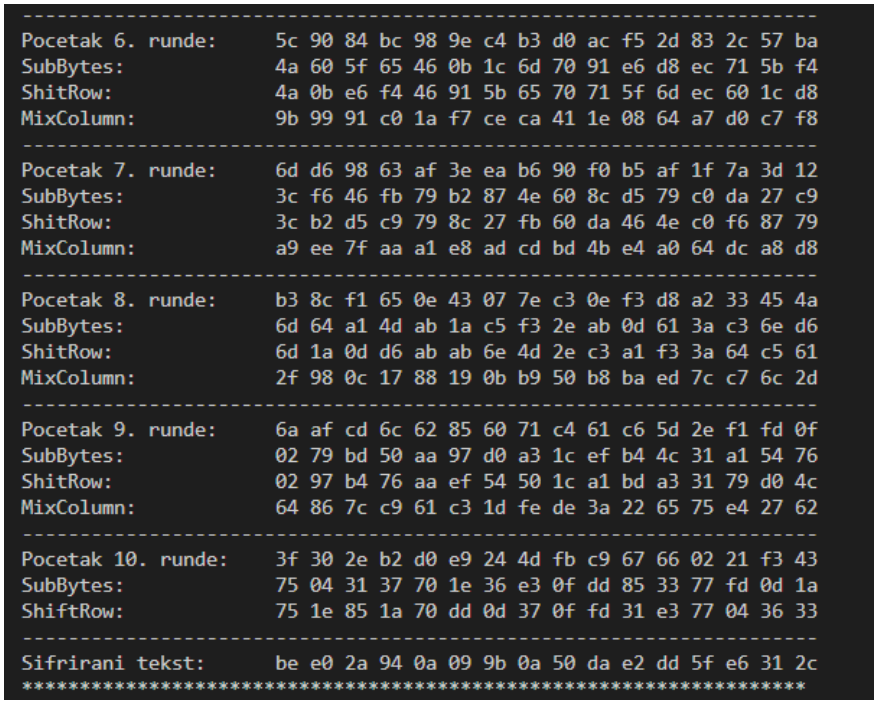
Kako bi mogli pokrenuti java program potrebna je *Test* klasa koja sadrži posebnu *main* metodu, ona se poziva prilikom pokretanja našeg porgrama. Prilikom pokretanja klase Test najprije je potrebno putem konzole unijeti tekst koji želimo šifrirati kao i ključ, te se vrši pretvaranje Stringa u niz bajtova kako bi tekst bio pogodan za daljnju obradu. U ovoj klasi vrši se ispisivanje rješenja dobivenih nakon procesa šifriranja i dešifriranja što je vidljivo na slici 6.18. Rezultat kompajlirana prikazan je slikama 6.19-6.24.

```
1  package Package;
2  import java.util.Scanner;
3
4  public class Test {
5      public static void main(String[] args) {
6
7      Scanner scanner = new Scanner(System.in);
8      System.out.print("Unesite tekst: ");
9      String inputStringText = scanner.nextLine();
10     byte[] tekst = inputStringText.getBytes();
11     System.out.print("Unesite kljuc: ");
12     String inputStringKljuc = scanner.nextLine();
13     byte[] kljuc = inputStringKljuc.getBytes();
14     scanner.close();
15
16     AES_sifriranje_desifriranje aes = new AES_sifriranje_desifriranje(kljuc, 4);
17     skeniranje.ispisNiza("Otvoreni tekst:      ", tekst);
18     skeniranje.ispisNiza("Kljuc:               ", kljuc);
19     byte[] izlaz1 = new byte[16];
20     aes.Sifriranje(tekst, izlaz1);
21     System.out.println("--------------------------------------------------------------------------");
22     skeniranje.ispisNiza("Sifrirani tekst:     ", izlaz1);
23
24     System.out.println("**************************************************************************");
25
26     skeniranje.ispisNiza("Sifrirani tekst:     ", izlaz1);
27     skeniranje.ispisNiza("Kljuc:               ", kljuc);
28     byte[] izlaz2 = new byte[16];
29     aes.deSifriranje(izlaz1, izlaz2);
30     System.out.println("--------------------------------------------------------------------------");
31     skeniranje.ispisNiza("Desifrirani tekst:   ", izlaz2);
32
33     }
34 }
```

Slika 6.18 *Klasa Test*

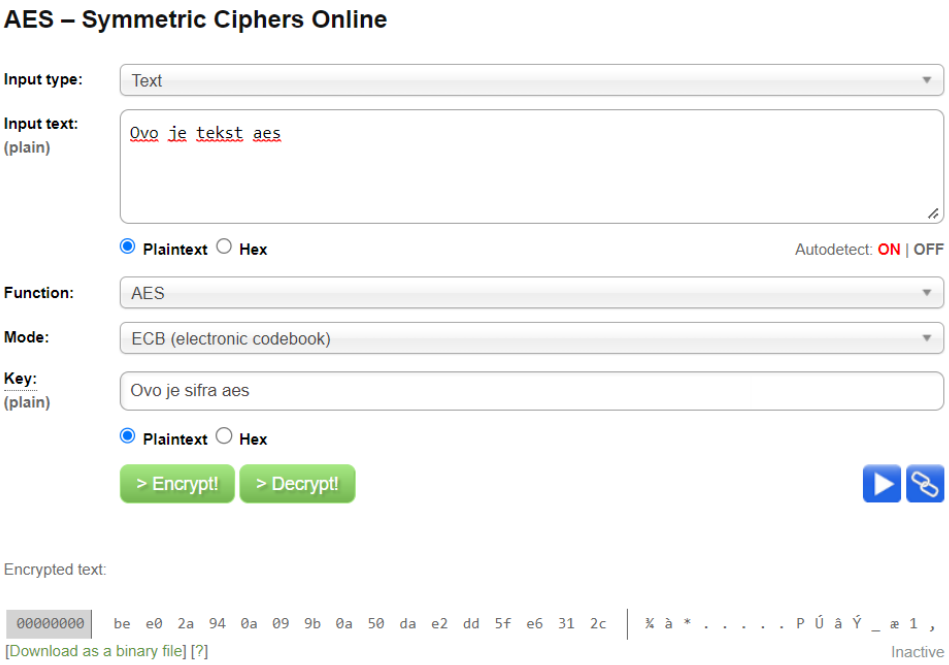


Slika 6.19 *Izlaz šifriranja 1/2*



Slika 6.20 *Izlaz šifriranja 2/2*

Za provjeru procesa šifriranja korišten je online kalkulator za AES[12]. Možemo zaključiti da se rezultati poklapaju.



Slika 6.21 *Online kalkulator rješenje šifriranja*

Slijede rezultati procesa dešifriranja.

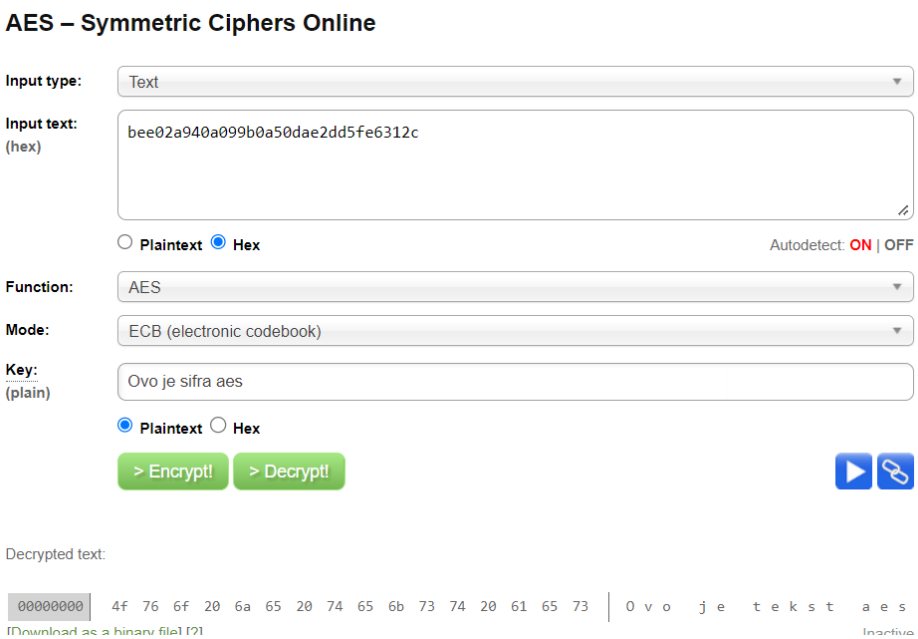
```
**********************************************************************
Sifrirani tekst:      be e0 2a 94 0a 09 9b 0a 50 da e2 dd 5f e6 31 2c
Kljuc:                4f 76 6f 20 6a 65 20 73 69 66 72 61 20 61 65 73
----------------------------------------------------------------------
Pocetak 1. runde:     75 1e 85 1a 70 dd 0d 37 0f fd 31 e3 77 04 36 33
invSbox:              75 04 31 37 70 1e 36 e3 0f dd 85 33 77 fd 0d 1a
invSubBytes:          3f 30 2e b2 d0 e9 24 4d fb c9 67 66 02 21 f3 43
invMixColumn:         02 97 b4 76 aa ef 54 50 1c a1 bd a3 31 79 d0 4c
----------------------------------------------------------------------
Pocetak 2. runde:     02 97 b4 76 aa ef 54 50 1c a1 bd a3 31 79 d0 4c
invSbox:              02 79 bd 50 aa 97 d0 a3 1c ef b4 4c 31 a1 54 76
invSubBytes:          6a af cd 6c 62 85 60 71 c4 61 c6 5d 2e f1 fd 0f
invMixColumn:         6d 1a 0d d6 ab ab 6e 4d 2e c3 a1 f3 3a 64 c5 61
----------------------------------------------------------------------
Pocetak 3. runde:     6d 1a 0d d6 ab ab 6e 4d 2e c3 a1 f3 3a 64 c5 61
invSbox:              6d 64 a1 4d ab 1a c5 f3 2e ab 0d 61 3a c3 6e d6
invSubBytes:          b3 8c f1 65 0e 43 07 7e c3 0e f3 d8 a2 33 45 4a
invMixColumn:         3c b2 d5 c9 79 8c 27 fb 60 da 46 4e c0 f6 87 79
----------------------------------------------------------------------
Pocetak 4. runde:     3c b2 d5 c9 79 8c 27 fb 60 da 46 4e c0 f6 87 79
invSbox:              3c f6 46 fb 79 b2 87 4e 60 8c d5 79 c0 da 27 c9
invSubBytes:          6d d6 98 63 af 3e ea b6 90 f0 b5 af 1f 7a 3d 12
invMixColumn:         4a 0b e6 f4 46 91 5b 65 70 71 5f 6d ec 60 1c d8
----------------------------------------------------------------------
Pocetak 5. runde:     4a 0b e6 f4 46 91 5b 65 70 71 5f 6d ec 60 1c d8
invSbox:              4a 60 5f 65 46 0b 1c 6d 70 91 e6 d8 ec 71 5b f4
invSubBytes:          5c 90 84 bc 98 9e c4 b3 d0 ac f5 2d 83 2c 57 ba
invMixColumn:         6a a6 7b cf ce 6a 70 92 5c c1 0d 59 71 38 99 d9
----------------------------------------------------------------------
```

Slika 6.22 *Izlaz dešifriranja 1/2*

```
------------------------------------------------------------------
Pocetak 6. runde:     6a a6 7b cf ce 6a 70 92 5c c1 0d 59 71 38 99 d9
invSbox:              6a 38 0d 92 ce a6 99 59 5c 6a 7b d9 71 c1 70 cf
invSubBytes:          58 76 f3 74 ec c5 f9 15 a7 58 03 e5 2c dd d0 5f
invMixColumn:         40 15 9a 5b 55 fc eb 3c c3 c3 72 31 16 ac 89 6b
------------------------------------------------------------------
Pocetak 7. runde:     40 15 9a 5b 55 fc eb 3c c3 c3 72 31 16 ac 89 6b
invSbox:              40 ac 72 3c 55 15 89 31 c3 fc 9a 6b 16 c3 eb 5b
invSubBytes:          72 aa 1e 6d ed 2f f2 2e 33 55 37 05 ff 33 3c 57
invMixColumn:         77 8e 1a 04 a0 13 81 aa d3 ad 62 a9 fe 86 d0 73
------------------------------------------------------------------
Pocetak 8. runde:     77 8e 1a 04 a0 13 81 aa d3 ad 62 a9 fe 86 d0 73
invSbox:              77 86 62 aa a0 8e d0 a9 d3 13 1a 73 fe ad 81 04
invSubBytes:          02 dc ab 62 47 e6 60 b7 a9 82 43 8f 0c 18 91 30
invMixColumn:         c1 2f 7a f8 a8 d0 2d e9 5a 63 7a c3 b9 b6 f0 d7
------------------------------------------------------------------
Pocetak 9. runde:     c1 2f 7a f8 a8 d0 2d e9 5a 63 7a c3 b9 b6 f0 d7
invSbox:              c1 b6 7a e9 a8 2f f0 c3 5a d0 7a d7 b9 63 2d f8
invSubBytes:          dd 79 bd eb 6f 4e 17 33 46 60 bd 0d db 00 fa e1
invMixColumn:         63 63 7c 63 63 d7 63 63 fe 63 63 c5 63 63 63 59
------------------------------------------------------------------
Pocetak 10. runde:    63 63 7c 63 63 d7 63 63 fe 63 63 c5 63 63 63 59
invSbox:              63 63 63 63 63 63 63 c5 fe d7 7c 59 63 63 63 63
invSubBytes:          00 00 00 00 00 00 00 07 0c 0d 01 15 00 00 00 00
------------------------------------------------------------------
Desifrirani tekst:    4f 76 6f 20 6a 65 20 74 65 6b 73 74 20 61 65 73
```

Slika 6.23 *Izlaz dešifriranja 2/2*

Korištenjem istog kalkulatora provjeren je rezultat procesa dešifriranja. Iz rezultata zaključujemo da je programsko rješenje ispravno jer se rezultati poklapaju u postupku šifriranja i postupku dešifriranja.



Slika 6.24 *Online kalkulator rješenje dešifriranja*

U usporedbi sa većinom ostalih implementiranih rješenja koje možemo pronaći na internetu ovo rješenje ne koristi gotove tablice potrebne za implementaciju već su eksplicitno izračunate vrijednosti pomoću koda. Prilikom implementacije cilj je bio proizvesti jasan i jednostavan Java program kako bi na konto objašnjene teorije i običnom čovjeku moglo biti jasno što kod u kojem dijelu radi. U budućem radu moguće je proširiti duljinu ključa uz modifikaciju KeyExpansion metode kako bi napravili još sigurnije programsko rješenje.

7. ZAKLJUČAK

Svjedoci smo sve većeg rasta korištenja Interneta i računalne mreže. Samim time mi kao korisnici možemo reći da se dobar dio našeg života vrti oko računala i upotrebe Interneta. Počevši od e-maila, poruka, društvenih mreža pa sve do platforme e-građani na kojoj su pohranjene sve bitne informacije o svakom čovjeku. Ako uzmemo u obzir informaciju da se na spomenutoj platformi nalaze važni dokumenti poput domovnice, rodnog lista, našeg zdravstvenog stanja i drugih bitnih informacija moramo biti svjesni da kao korisnici moramo postupati pažljivo i učiniti sve što je u našoj moći da se zaštitimo. Iz toga razloga je znanje o računalnoj sigurnosti sve neophodnije kako bi što uspješnije mogli zaštiti svoju privatnost. Enkripcijski algoritmi igraju vitalnu ulogu u zaštiti izvornih podataka od neovlaštenog pristupa. Postoje različite vrste algoritama za šifriranje podataka. Algoritam *Advanced Encryption Standard* (AES) trenutno je jedan od najboljih algoritama za zaštitu informacija i široko je podržan i usvojen na hardveru i softveru. Ovaj algoritam može se nositi s različitim veličinama ključeva kao što su 128, 192 i 256 bita sa 128-bitnom blok šifrom u ovisnosti o potrebama korisnika. U ovom radu objašnjene su brojne važne značajke AES algoritma kao i njena matematička pozadina koja predstavlja ključan dio ovoga algoritma. Iz rada možemo zaključiti da teorija prstena ima važnu ulogu u stvarnom svijetu, a algebarska struktura koju nudi čini računarstvo učinkovitijim. Sada možemo zamisliti da je slovo predstavljeno različitim polinomom, a također je predstavljena zanimljiva veza između matričnog množenja i modularnog množenja s fiksnim polinomom. Općenito, AES standard uvelike koristi matematičke koncepte za pružanje velike sigurnosti u enkripciji, a istovremeno nudi izvrsne performanse. Ovaj rad je također fokusiran na korištenje 128-bitnog ključa što je prikazano u samoj implementaciji algoritma. Ukoliko bismo htjeli koristiti 192-bitni ili 256-bitni ključ, jedina velika razlika je u tome što bismo morali napraviti 12, odnosno 14 krugova. Ovo bi također modificiralo korak proširenja ključa. Što se više rundi algoritma izvrši time se mijenjaju performanse za sigurnost. Iako se AES smatra sigurnim jer nije imao velike napade, dolazimo do pitanja kako će se on nositi sa novim tehnologijama koje se iz dana u dan razvijaju.

I. Litaratura

[1] Federal Information Processing Standards Publication, DATA ENCRYPTION STANDARD (DES), U.S. DEPARTMENT OF COMMERCE/National Institute of Standards and Technology, 1999.

[2] J. Daemen, V. Rijmen, The Design of Rijndael: The Advanced Encryption Standard (AES) Second Edition, Springer, 2021.

[3] C. Paar, J. Pelzl, Understanding Cryptography: A Textbook for Students and Practitioners, Springer, 2010.

[4] Federal Information Processing Standards Publication, ADVANCED ENCRYPTION STANDARD (AES), National Institute of Standards and Technology (NIST), 2001.

[5] C. Cid, S. Murphy, M. Robshaw, Algebraic Aspects of the Advanced Encryption Standard, Springer, 2006.

[6] S. Krešić-Jurić, Algebarske strukture: Skripta, Odjel za matematiku Prirodoslovno-matematički fakultet Split, 2013.

[7] Djeljivost. Kongruencije. Konačna polja, Fakultet elektrotehnike i računarstva, Zagreb

Dostupno na: https://www.fer.unizg.hr/_download/repository/diskont1-11.pdf

[8] W. Stallings, Cryptography and Network Security Principles and Practices, Fourth Edition, Prentice Hall, 2005.

[9] A. Kak, Computer and Network Security: Lecture 7- Finite Fields, Purdue University, 2022.

[10] C. Dong, Math in Network Security: A Crash Course,

Dostupno na: https://www.doc.ic.ac.uk/~mrh/330tutor/index.html

[11] R. Mardisalu, Advanced Encryption Standard (AES), 2020.,

Dostupno na: https://thebestvpn.com/advanced-encryption-standard-aes/#attacks

[12] Online kalkulator AES dostupan na: http://aes.online-domain-tools.com/

II. Sažetak

National Institute of Standards and Technology (NIST) je 1997. godine raspisao natječaj za novi napredni standard šifriranja, te je 2000. godine objavio da je odabrao Rijndaela kao AES. Rijndaela je naslijedio DES koji je postajao sve ranjiviji. Advanced Encryption Standard je 2001. godine odobren kao moderni standard za šifriranje podataka. Ima široku primjenu na raznim područjima, od povjerljivih državnih dokumenata do prometa putem Wi-Fi usmjerivača. AES koji koristimo su razvila dva belgijska kriptografa Joan Daemen i Vincent Rijmen. Razvili su šifru Rijandel kojoj veličina bloka varira od 128, 192 do 256 bita. Međutim AES koristi samo veličinu bloka od 128 bita. Vrlo bitno svojstvo AES-a je što koristi isti ključ za proces šifriranja i dešifriranja. Rijndael podržava tri duljine ključa 128, 192 i 256 bita. U procesu šifriranja i dešifriranja razlikujemo tri vrste slojeva: Sloj dodavanja ključa (eng. Key Addition layer), Sloj zamjene bajtova (eng. Byte Substitution layer (S-Box)) i Difuzijski sloj (eng. *Diffusion layer*) koji se dijeli na dva podsloja (ShiftRow i MixColumn). U pozadini tih slojeva koristi se nekoliko ključnih matematičkih koncepata iz linearne algebre i teorije prstena. Algoritam uključuje 10 do 14 rundi u kojima se izmjenjuju navedeni slojevi s manjim razlikama u prvoj i zadnjoj rundi izvođenja. U svakom od navedenih koraka AES algoritma vrši se jednostavna radnja s matematičkim izračunima.

Ključne riječi: kriptografija, *National Institute of Standards and Technology*, NIST, Rijndaela, Advanced Encryption Standard, AES, DES, šifriranje, deširiranje, ključ, Key Addition layer, Byte Substitution layer, *Diffusion layer,* ShiftRow, MixColumn, linearna algebra, grupa, prsten, modularna aritmetika, polje, Galois polje

III. Summary

In 1997, the National Institute of Standards and Technology (NIST) announced a competition for a new advanced encryption standard, and in 2000 announced that it had selected Rijndael as AES. Rijndael was succeeded by DES, which was becoming increasingly vulnerable. The Advanced Encryption Standard was approved in 2001 as the modern standard for data encryption. It is widely used in a variety of areas, from classified government documents to traffic through Wi-Fi routers. The AES we use was developed by two Belgian cryptographers, Joan Daemen and Vincent Rijmen. They developed the Rijandel cipher, whose block size varies from 128, 192 to 256 bits. However, AES only uses a block size of 128 bits. A very important property of AES is that it uses the same key for the encryption and decryption process. Rijndael supports three key lengths 128, 192 and 256 bits. In the encryption and decryption process, we distinguish three types of layers: Key Addition layer, Byte Substitution layer (S-Box) and Diffusion layer, which is divided into two sublayers (ShiftRow and MixColumn). In the background of these layers, several key mathematical concepts from linear algebra and ring theory are used. The algorithm includes 10 to 14 rounds in which the specified layers are alternated with minor differences in the first and last rounds of execution. In each of the mentioned steps of the AES algorithm, a simple action with mathematical calculations is performed.

Key words: cryptography, National Institute of Standards and Technology, NIST, Rijndaela, Advanced Encryption Standard, AES, DES, encryption, decryption, key, Key Addition layer, Byte Substitution layer, Diffusion layer, ShiftRow, MixColumn, linear algebra, group, ring, modular arithmetic, field, Galois field

IV. Popis ilustracija

SLIKE

TABLICE

V. Životopis

Dubravka Doko je rođena u Osijeku 03. siječnja 1998. godine. Odrasla je u Đakovu gdje je u razdoblju od 2004.-2012. godine stekla osnovnoškolsko obrazovanje. Nakon završetka osnovne škole upisuje Gimnaziju Antuna Gustava Matoša Đakovo, prirodoslovno matematički smjer. 2016. godine upisuje stručni studij Informatike na Fakultetu elektrotehnike računarstva i informacijskih tehnologija gdje je stekla akademski naziv stručna prvostupnica inženjerka elektrotehnike. Na istom fakultetu 2020. godine upisuje diplomski sveučilišni studij elektrotehnike, smjer Komunikacije i informatika, izborni blok Mrežne tehnologije