

Advanced Encryption Standard - vodič i implementacija

autor: **Srđan Rašić**
srdan.rasic@gmail.com

travanj 2012, v1.1

Promjene u verziji 1.1:

- ispravljena greška u kodu funkcije kriptiraj
- dodani ispitni vektori
- dodana napomena o problemu posmaka udesno

Uvod

Kratika *AES* podrazumijeva specifikaciju za kriptiranje elektroničkih podataka, no uobičajeno je tom kraticom nazivati i sâm algoritam kriptiranja. Algoritam koji se koristi je inačica *Rijndael* (izgovara se *Rain Doll*) algoritma.

Veličina bloka kojeg se kriptira po *AES* standardu učvršćena je na 128 bitova (odnosno 16 okteta), dok ključ kriptiranja može biti veličine 128, 192 ili 256 bitova (odnosno 16, 24 ili 32 okteta). Iako ključ nema teoretskog maksimuma, ovo su standardne veličine. *Rijndael* postupak dozvoljava i blokove različitih veličina, no kako *AES* specifikacija blok učvršćuje na 16 okteta, ovaj vodič opisuje samo tu standardnu inačicu *AES* postupka.

AES kriptiranje svodi se na uzastopnu primjenu određenih transformacija koje ulazni, tj. jasni tekst pretvaraju u izlazni, tj. kriptirani tekst. Pod pojmom tekst u ovom slučaju podrazumijeva se blok podataka od 16 okteta (128 bitova). Primjena transformacija odvija se u rundama (krugovima). Svaka runda sastoji se od nekoliko koraka od kojih jedan ovisi o ključu kriptiranja. U slučaju dekriptiranja obavlja se proces primjene inverznih transformacija.

Kriptiranje se može opisati funkcijom

$$C = f(P, K)$$

gdje P predstavlja jasni (ulazni) tekst, K ključ kriptiranja, a C kriptirani tekst.

Zapis podataka u računalnom memoriji

Blok podataka (jasni tekst) vizualno se prikazuje kao 4×4 matrica i naziva se *stanje*. Ključ također vizualno prikazujemo pomoću matrice $4 \times n$ gdje n odgovara veličini ključa u riječima¹ (za potključ, koji će biti opisan kasnije, uvijek vrijedi $n = 4$). Okteti teksta ili ključa u matricu se zapisuju stupac po stupac - prvo se popunjava prvi stupac, pa drugi itd. Prilikom implementacije jednostavnije je matrice *odmotati* u nizove okteta (konkateniranjem redova ili stupaca). Implementacija u ovom radu konkatenira stupce (jednostavnije je). Na primjer, izvorni tekst/ključ 2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c matrično prikazujemo ovako:

¹ riječ = skupina od četiri okteta, 32-bitni tip podatka

```
2b 28 ab 09
7e ae f7 cf
15 d2 15 4f
16 a6 88 3c
```

Tu matricu u računalnoj memoriji zapisujemo u jednodimenzionalni niz (polje) okteta odmotavanjem matrice po stupcima čime dobivamo niz s elementima 2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c. Redoslijed okteta u nizu zapravo je jednak redoslijedu okteta u jasnom tekstu, odnosno ključu kriptiranja.

Prilikom pisanja pseudokodova korišten je oktet kao osnovni tip podatka radi lakšeg prepisivanja u C ili sličan jezik. Također, valja pripaziti na obavljanje operacije posmaka udesno. Ako se radi o *signed* tipu podatka posmak udesno obavlja se tako da se najviši bitovi popunjavaju onim bitom koji je prije operacije bio najviši. Kod *unsigned* tipa podatka prilikom posmaka udesno uvijek se dodaju nule stoga je preporučljivo koristiti *unsigned* tip podatka kao što je npr. *unsigned char* u C-u. Ako nemate mogućnost korištenja *unsigned* tipa podatka potreban je dodatan oprez. Iz tog razloga Java pruža operator `>>>` koji obavlja posmak udesno i pritom najviše bitove popunjava nulama.

Postupka kriptiranja

Postupak najbolje možemo shvatiti ako prvo pogledamo njegov *grubi* opis. Prirodno ga je podijeliti u dvije faze. Prva faza podrazumijeva postupak proširenja izvornog ključa kriptiranja kojim se stvara određeni broj potključeva, a druga faza obuhvaća obavljanje transformacija u rundama nad blokom podataka kojeg se kriptira pri čemu se u svakoj rundi koristi drugi potključ. Postupak izgleda ovako:

1. Proširenje ključa kriptiranja (stvaranje potključeva)
2. Inicijalna runda (ne broji se kao runda)
 - a) Dodaj potključ
3. Runde (ponavljaj *brojRundi-1* puta)
 - a) Zamijeni oktete
 - b) Posmakni redove
 - c) Pomiješaj stupce
 - d) Dodaj potključ

4. Posljednja runda

- a) Zamijeni oktete
- b) Posmakni redove
- d) Dodaj potključ

Broj rundi *brojRundi* je predodređen i ovisi o veličini izvornog ključa kriptiranja:

Za ključ veličine **16** okteta izvodi se **10** rundi.

Za ključ veličine **24** okteta izvodi se **12** rundi.

Za ključ veličine **32** okteta izvodi se **14** rundi.

Proširenje ključa kriptiranja

Ovim postupkom izvorni ključ se proširuje u veći, prošireni, ključ. Dijelove tog proširenog ključa nazivamo potključevima. U svakoj rundi kriptiranja koristi se drugi potključ. Veličina proširenog ključa ovisi o veličini izvornog ključa. Odredimo tu veličinu intuitivno; Neka je izvorni ključ veličine 16 okteta. To znači da je potrebno obaviti deset rundi. U svakoj od tih deset rundi dodaje se potključ, a uz to potključ se dodaje i tijekom inicijalizacije (inicijalne runde). Kako svako dodavanje potključa podrazumijeva dodavanje različitog potključa, potrebno nam je $10 + 1$ potključeva. Veličina potključa mora biti jednaka veličini bloka koji se kriptira (bit će jasno kasnije), dakle 16 okteta jer opisujemo standardnu inačicu *AES-a*. Iz toga proizlazi da je veličina proširenog ključa $16 * (10 + 1)$ okteta ukoliko je izvorni ključ veličine 16 okteta (zbog čega imamo 10 rundi). Opća formula sada je očevidna:

$$veličinaProširenogKljuča = (brojRundi + 1) * veličinaBloka$$

Jezgra proširenja

Proširenje izvornog ključa sastoji se od primjene nekih operacija od kojih osnovnu operaciju nazivamo *jezgraProširenja* (od *engl.* Key schedule core). No da bi se nju shvatilo potrebno je prvo uvesti neke osnovne transformacije.

Rotiraj

Transformacija koja posmiče riječ za jedan oktet ulijevo. Primjer:

```
rotiraj( 1A 20 21 FF ) = 20 21 FF 1A
```

U implementaciji uvodimo dodatni argument *korak* koji nam omogućuje da primijenimo transformaciju nad stupcem ili nad retkom matrice. Obzirom da smo matricu odmotali u niz po stupcima, ako postavimo *korak* na 1, transformacija će se primijeniti nad stupcem matrice kojemu se prvi element nalazi na adresi *r*. Uz *korak* 4 *skakutali* bismo po nizu svaka četiri okteta počevši od okteta na adresi *r*, a to je analogno iteriranju retka matrice.

```
rotiraj (byte[] r, int korak):
    byte b = r[0];
    za i = 0; i < 3; i++ čini:
        r[i*korak] = r[(i+1)*korak];
    r[3*korak] = b;
```

Rcon

rcon(i) je funkcija zadana u *Rijndael* konačnom polju. Sve potrebne vrijednosti mogu se izračunati unaprijed pa se ova transformacija svodi na obično indeksiranje niza. Primjer:

```
rcon[0x01] = 0x01
```

To je dakle vrijednost elementa indeksa 1 niza *rcon*. Pogledaj *Dodatak A*!

S-Box (S-kutija)

S-Box ili *S-kutija* zapravo je tablica koja se koristi prilikom zamjene okteta. Ona je oblika:

		0	1	2	3	...	f
---		--		--		--	
00		63	7c	77	7b	...	67
10		ca	82	c9	7d	...	c0
20		b7	fd	93	26	...	15
..	

Zamjenski oktet dobiva se tako da se traži redak kojeg određuje prva polovica (*nibble*) izvornog okteta i stupac kojeg određuje druga polovica izvornog okteta. Recimo da tražimo

zamjenski oktet okteta $0x1f$. Prva polovica je $0x10$, a druga $0x0f$. Stoga je zamjenski oktet onaj u retku $0x10$ i stupcu $0x0f$, dakle $0xc0$. Izračun tablice dosta je kompliciran, no za naše potrebe dovoljno je koristiti već izračunatu tablicu. Tablicu spremamo u obliku niza pa se određivanje zamjenskog okteta svodi na indeksiranje niza. Primjer:

$$sbox[0x02] = 0x77$$

Dakle, izvorni oktet $0x02$ predstavlja indeks na kojemu se nalazi zamjenski oktet $0x77$.

Pogledaj *Dodatak A*!

Ove tri transformacije nam omogućuju da definiramo operaciju *jezgraProširenja* koja će biti korištena u postupku proširenja izvornog ključa. To je vrlo jednostavna operacija koja se svodi na primjenu triju upravo definiranih transformacija:

```
jezgraProširenja (byte[4] r, int iteracija):  
    rotiraj(r, 1);  
    za i = 0; i < 4; i++ čini:  
        r[i] = sbox[r[i]];  
    r[0] = r[0] xor rcon[iteracija];  
    vrati r;
```

Prvo se riječ *r* posmakne za jedan oktet, zatim se svaki oktet zamijeni s odgovarajućim iz *s-kutije* i na kraju se primjeni operacija *isključivo* ili na prvi oktet riječi *r* i oktet *rcon* transformacije nad brojem iteracije.

Ovime su definirane sve potrebne operacije za postupak proširenja ključa.

Postupak proširenja ključa

Neka je *n* veličina izvornog ključa u riječima, a *b* veličina proširenog ključa u riječima, tj. $n \in \{4, 6, 8\}$ i $b \in \{176/4=44, 208/4=52, 240/4=60\}$. Vrijednosti *b* dobivamo po formuli za veličinu proširenog ključa koja je definirana na početku. Postupak je opisan sljedećim koracima:

- I. Prvih *n* riječi proširenog ključa se dobije tako da se kopira izvorni ključ
- II. Broj iteracije se postavlja na 1, tj. *iteracija* = 1
- III. Dok god nismo proširili ključ do *b* riječi, radi sljedeće čime proširuješ ključ za *n* riječi:

a) Prva riječ stvara se na sljedeći način:

1. Neka je t varijabla tipa *riječ*, a k indeks zadnje stvorene riječi proš. ključa
2. Varijabli t pridružimo riječ s indeksom k
3. Primijenimo operaciju *jezgraProširenja* nad t
4. Povećamo broj iteracije, $iteracija++$
5. Primijenimo operaciju *isključivo ili* nad riječi t i riječi s indeksom $k - (n - 1)$
6. Rezultat je riječ kojom proširujemo ključ

b) Iduće tri riječi stvaraju se na sljedeći način (ponavljaj ovo tri puta):

1. Neka je t varijabla tipa *riječ*, a k indeks zadnje stvorene riječi proš. ključa
2. Varijabli t pridružimo riječ s indeksom k
3. Primijenimo operaciju *isključivo ili* nad riječi t i riječi s indeksom $k - (n - 1)$
4. Rezultat je riječ kojom proširujemo ključ

c) Ako je ključ 256-bitni ($n = 8$) stvaramo još jednu riječ ovako:

1. Neka je t varijabla tipa *riječ*, a k indeks zadnje stvorene riječi proš. ključa
2. Varijabli t pridružimo riječ s indeksom k
3. Zamijenimo sva četiri okteta riječi t s odgovarajućim oktetima iz *s-kutije*
4. Primijenimo operaciju *isključivo ili* nad riječi t i riječi s indeksom $k - (n - 1)$
5. Rezultat je riječ kojom proširujemo ključ

d) Ako je ključ 256-bitni izvodimo sljedeće tri puta, a ako je 192-bitni, dva puta:

1. Neka je t varijabla tipa *riječ*, a k indeks zadnje stvorene riječi proš. ključa
2. Varijabli t pridružimo riječ s indeksom k
3. Primijenimo operaciju *isključivo ili* nad riječi t i riječi s indeksom $k - (n - 1)$
4. Rezultat je riječ kojom proširujemo ključ

Vidljivo je da je b djeljiv s n čime je predviđeno da se korak *III.* izvrši u potpunosti $b / n - 1$ puta. Važno je napomenuti da se koraci c) i d) ne izvode ukoliko je ključ 128-bitni i da se korak c) ne izvodi ni ukoliko je ključ 192-bitni.

Iz opisanog postupka vrlo lako možemo napisati i funkciju koja proširuje ključ. Uočimo da se dosta operacija u različitim koracima ponavlja. U svakom koraku (*a...d*) uvijek se na početku varijabla t postavlja na prethodno stvorenu riječ, a na kraju se primjenjuje operacija

isključivo ili. Nadalje, operacija *jezgraProširenja* primjenjuje se kad god stvorimo n riječi, dok se zamjena okteta s odgovarajućima iz *s-kutije* obavlja samo u slučaju 256-bitnog izvornog ključa te ako smo do sad stvorili 4 riječi (jednu u koraku *a* i tri u koraku *b*). Zbog tih svojstava moguće je izmijeniti postupak čime ćemo izbjeći ponavljanje operacija. Umjesto oblika

```
dok trenutnaVeličina < veličinaProširenogKljuča čini:
    jezgraProširenja(r);
    za i = 0; i < 4; i++ čini:
        neka_operacija();
```

mi ćemo koristiti ovaj

```
dok trenutnaVeličina < veličinaProširenogKljuča čini:
    ako veličinaProširenogKljuča mod veličinaKljuča == 0:
        jezgraProširenja(r);
    neka_operacija();
```

Oblik se svodi na istu stvar, ali omogućuje nam da izbjegnemo nepotrebno dupliciranje koraka.

```
enum VeličinaKljuča { AES_128 = 16, AES_192 = 24, AES_256 = 32 }

proširiKljuč (byte[n] izvorniKljuč, VeličinaKljuča n):
    int trVel = 0;           /* trenutna veličina proš. ključa */
    int iteracija = 1;       /* trenutna rcon iteracija (II. korak) */
    int b;                   /* veličina proširenog ključa */
    byte[4] t;               /* pomoćna riječ */

    /* Odredimo veličinu proširenog ključa u oktetima */
    switch (n):
        case AES_128:
            b = (10 + 1) * 16; break;
        case AES_192:
            b = (12 + 1) * 16; break;
        case AES_256:
            b = (14 + 1) * 16; break;

    byte[b] ključ;           /* prošireni ključ */

    /* I. korak */
    za i = 0; i < n; i++ čini:
        ključ[i] = izvorniKljuč[i];
    trVel += n;
```



```

/* III. korak */
dok trVel < b čini:
    /* Varijabla t = zadnja stvorena riječ */
    za i = 0; i < 4; i++ čini:
        t[i] = ključ[(trVel - 4) + i];

    /* Svakih 16, 24 ili 32 okteta primjenimo
       jezgruProširenja i povećamo # iteracija */
    ako trVel mod n == 0:
        t = jezgraProširenja(t, iteracija++);

    /* Za 256-bitne ključeve imamo još jednu
       operaciju zamjene iz s-kutije */
    ako n == AES_256 i (trVel mod n == 16):
        za i = 0; i < 4; i++ čini:
            t[i] = sbbox[t[i]];

    /* Primjenimo operaciju xor nad riječi t i riječi... */
    za i = 0; i < 4; i++ čini:
        ključ[trVel] = ključ[trVel - n] xor t[i];
        trVel++;

vrati ključ;

```

Kriptiranje bloka podataka

Kriptiranje bloka podataka ugrubo je opisano na prvoj stranici ovog rada. Kriptiranje se sastoji od uzastopne primjene određenih transformacija organiziranih po rundama. Transformacije se obavljaju nad blokom podataka koji se kriptira (jasnim tekstom).

Opišimo za početak transformacije.

Dodavanje potključa

Ova transformacija predstavlja kombiniranje stanja (bloka podataka) s potključem. Svodi se na primjenu operacije *isključivo* ili između svakog odgovarajućeg okteta stanja i okteta potključa. Formalno to znači

$$S_{ij} = S_{ij} \text{ xor } P_{ij}$$

gdje S_{ij} predstavlja oktet (element) retka i i stupca j matrice stanja (bloka podataka), a P_{ij} oktet (element) retka i i stupca j matrice potključa. Iz toga prozlaži da S i P moraju biti istih dimenzija čime ograničavamo veličinu potključa na 4×4 okteta.

Pseudokod te transformacije izgleda ovako (matrice su *odmotane* u nizove):

```
dodajPotključ (byte[16] stanje, byte[16] potključ):
    za i = 0; i < 16; i++ čini:
        stanje[i] = stanje[i] xor potključ[i];
```

Zamjena okteta

Vrlo jednostavna transformacija. Svaki oktet matrice stanja zamjenjuje se s odgovarajućim oktetom iz *S-kutije* koja je opisana ranije.

```
zamijeniOktete (byte[16] stanje):
    za i = 0; i < 16; i++ čini:
        stanje[i] = sbox[stanje[i]];
```

Posmicanje redova

Transformacija se izvodi nad redovima matrice stanja. Ciklički se okteti retka posmiču ulijevo za određeni pomak. U slučaju 128-bitnog bloka podataka s kakvim mi radimo, pomaci su definirani ovako; Prvi redak ostaje nepromijenjen. Drugi redak se posmiče za jedan oktet, treći za dva okteta, a četvrti za tri okteta.

Za obavljanje ove transformacije možemo iskorisiti ranije definiranu transformaciju *rotiraj* koja posmiče riječ (a redak matrice 4×4 jest riječ) za jedan oktet ulijevo.

```
posmakniRedove (byte[16] stanje):
    za i = 1; i < 4; i++ čini:
        za j = 0; j < i; j++ čini:
            rotiraj (stanje[i], 4);
```

Miješanje stupaca

Svaki stupac matrice stanja promatra se kao polinom u tzv. Galoisovom polju $GF(2^8)$ te se množi polinomom $c(x) = 3x^4 + x^2 + x + 2$ modulo fiksni ireducibilni polinom $g(x) = x^8 + x^4 + x^3 + x + 1$ (*0x11b*). Operacija se može prikazati u matričnom obliku

$$\begin{bmatrix} r_0 \\ r_1 \\ r_2 \\ r_3 \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

gdje a_i predstavlja stupac nad kojim se obavlja transformacija, a r_i rezultat. U implementaciji ta operacija može se pojednostaviti tako da se množenje s 2 zamijeni posmakom ulijevo i, ako je došlo do preljeva, isključivim ili s $0x1b$, odnosno s već spomenutim fiksним ireducibilnim polinomom, jer rezultat mora ostati u polju $GF(2^8)$. Ako radimo s 8-bitnim tipom podatka, posmakom ulijevo najviši bit se gubi pa je dovoljno primijeniti isključivo ili s $0x1b$. Operacija množenja s 2 izvedena na taj način naziva se *xputa*. Množenje s 3 može se zamijeniti množenjem s 2 i isključivim ili, npr. $0x57 \cdot 0x03 = 0x57 \cdot (0x02 \oplus 0x01) = 0x57 \cdot 0x02 \oplus 0x57$.

```
pomiješajStupce (byte[16] stanje):
    for i = 0; i < 4; i++ čini:
        /* Matrica je odmotana u niz po stupcima, pa ćemo
           funkciji koja mijesha jedan stupac poslati pokazivač
           na prvi element stupca */
        pomiješajStupac(stanje[i*4]);

xputa (byte a):
    /* p je 1 ako je najviši bit od a[i] postavljen,
       inače je 0. p je zastavica koja označava hoće li
       doći do preljeva prilikom pomaka ulijevo */
    byte p = a rsh 7; /* rhs je pomak udesno */

    byte rez = a lhs 1; /* množenje s 2 je pomak ulijevo */

    /* ... ako je došlo preljeva (p == 1), obavi
       ovu operaciju da rezultat ostane u GF(28) polju. */
    ako p == 1:
        rez = rez xor 0x1b;

    vrati rez;

pomiješajStupac (byte[] s):
    byte[4] a;      /* Kopija ulaznog stupca */
    byte[4] b;      /* Svaki element iz a pomnožen s 2 u GF(28) */

    dok i = 0; i < 4; i++ čini:
        a[i] = s[i];
        b[i] = xputa (a[i]);

    /* a[n] xor b[n] je element n pomnožen s 3 u GF(28) */

    /*      2*a0      +a3      +a2      +2*a1      +a1 (tj. +3*a1) */
    s[0] = b[0] xor a[3] xor a[2] xor b[1] xor a[1];
    s[1] = b[1] xor a[0] xor a[3] xor b[2] xor a[2];
    s[2] = b[2] xor a[1] xor a[0] xor b[3] xor a[3];
    s[3] = b[3] xor a[2] xor a[1] xor b[0] xor a[0];
```

Kriptiranje bloka podataka

Opisavši sve potrebne transformacije imamo svo potrebno predznanje za shvaćanje postupka kodiranja po *AES* specifikaciji. Okvirni opis postupka dan je na prvoj stranici pa će ovdje biti prikazan samo pseudokod.

```
enum VeličinaKljuča { AES_128 = 16, AES_192 = 24, AES_256 = 32 }

kriptiraj (byte[16] blokPodataka, byte[n] ključ, VeličinaKljuča n):
    byte[] prošireniKljuč = proširiKljuč (ključ, n);
    byte[16] stanje = blokPodataka /* memcpy, clone i sl. */
    int brojRundi;
    int trRunda = 0;

    switch (n):
        case AES_128:
            brojRundi = 10; break;
        case AES_192:
            brojRundi = 12; break;
        case AES_256:
            brojRundi = 14; break;

    /* Inicijalna runda */
    dodajPotključ(blokPodataka, prošireniKljuč[trRunda]);

    /* Runde (sve osim zadnje) */
    za trRunda = 1; trRunda <= brojRundi - 1; trRunda++ čini:
        zamijeniOktete(blokPodataka);
        posmakniRedove(blokPodataka);
        pomiješajStupce(blokPodataka);
        /* Potključevi su veličine 16 okteta i složeni su
           jedan iza drugog u niz */
        dodajPotključ(blokPodataka, prošireniKljuč[trRunda*16]);

    /* Zadnja runda (ne sadrži miješanje stupaca) */
    zamijeniOktete(blokPodataka);
    posmakniRedove(blokPodataka);
    dodajPotključ(blokPodataka, prošireniKljuč[trRunda*16]);

    vrati stanje; /* Kriptirani blok podataka */
```

Svoju implementaciju možete ispitati pomoću ispitnih vektora danih u *dodatku B*.

Dekriptiranje bloka podataka

Dekriptiranje se svodi na postupak inverzan postupku kriptiranja. Redoslijed kojim se obavljaju runde je identičan onome prilikom kriptiranja, no transformacije unutar rundi primjenjuju se drugačijim redoslijedom. Transformacije u ovom postupku inverzne su onima koje se koriste u kriptiranju. Transformacija *dodajPotključ* sama je sebi inverz te ju nije potrebno modificirati. Inverzna transformacija zamjene okteta implementira se korištenjem inverzne *s-kutije* čije su vrijednosti dane u dodatku A i naziva se *invZamijeniOktete*. Inverz transformaciju posmicanja redova *invPosmakniRedove* također je vrlo jednostavno napisati (ne posmiče se ulijevo već udesno), dok je inverz miješanja stupaca *invPomiješajStupce* nešto složenije. U tom slučaju svaki stupac matrice *stanje* množi se s inverznim polinomom koji je oblika $c^{-1}(x) = 11x^3 + 13x^2 + 9x + 14$. Matrično to izgleda ovako:

$$\begin{bmatrix} r_0 \\ r_1 \\ r_2 \\ r_3 \end{bmatrix} = \begin{bmatrix} 14 & 11 & 13 & 9 \\ 9 & 14 & 11 & 13 \\ 13 & 9 & 14 & 11 \\ 11 & 13 & 9 & 14 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

Računanje ovog sustava može se svesti na postupak analogan postupku u *pomiješajStupce*. Pogledaj str. 18 specifikacije Rijndael algoritma². Još preostaje definirati redoslijed izvođenja transformacija koji izgleda ovako:

1. Inicijalna runda
 - a) *dodajPotključ*
3. Runde (ponavljaj *brojRundi-1* puta)
 - a) *invPosmakniRedove*
 - b) *invZamijeniOktete*
 - c) *dodajPotključ*
 - d) *invPomiješajStupce*
4. Posljednja runda
 - a) *invPosmakniRedove*

² A Specification for Rijndael, the AES Algorithm, Dr. Brian Gladman, v3.11, 12th Sept 2003
<http://asmaes.sourceforge.net/rijndael/rijndaelImplementation.pdf>

- b) *invZamijeniOktete*
- d) *dodajPotključ*

Dodatak A: Izračunate vrijednosti funkcija

Prilikom implementacije preporučljivo je koristiti već izračunate vrijednosti ovih triju funkcija ukoliko memorijski resursi nisu ograničeni.

Rcon

```
rcon[256] = {  
0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a,  
0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39,  
0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a,  
0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8,  
0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef,  
0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc,  
0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b,  
0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3,  
0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94,  
0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20,  
0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35,  
0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f,  
0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04,  
0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63,  
0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd,  
0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d }
```

S-Box

```
sbox[256] = {  
0x63, 0x7C, 0x77, 0x7B, 0xF2, 0x6B, 0x6F, 0xC5, 0x30, 0x01, 0x67, 0x2B, 0xFE, 0xD7, 0xAB, 0x76,  
0xCA, 0x82, 0xC9, 0x7D, 0xFA, 0x59, 0x47, 0xF0, 0xAD, 0xD4, 0xA2, 0xAF, 0x9C, 0xA4, 0x72, 0xC0,  
0xB7, 0xFD, 0x93, 0x26, 0x36, 0x3F, 0xF7, 0xCC, 0x34, 0xA5, 0xE5, 0xF1, 0x71, 0xD8, 0x31, 0x15,  
0x04, 0xC7, 0x23, 0xC3, 0x18, 0x96, 0x05, 0x9A, 0x07, 0x12, 0x80, 0xE2, 0xEB, 0x27, 0xB2, 0x75,  
0x09, 0x83, 0x2C, 0x1A, 0x1B, 0x6E, 0x5A, 0xA0, 0x52, 0x3B, 0xD6, 0xB3, 0x29, 0xE3, 0x2F, 0x84,  
0x53, 0xD1, 0x00, 0xED, 0x20, 0xFC, 0xB1, 0x5B, 0x6A, 0xCB, 0xBE, 0x39, 0x4A, 0x4C, 0x58, 0xCF,  
0xD0, 0xEF, 0xAA, 0xFB, 0x43, 0x4D, 0x33, 0x85, 0x45, 0xF9, 0x02, 0x7F, 0x50, 0x3C, 0x9F, 0xA8,  
0x51, 0xA3, 0x40, 0x8F, 0x92, 0x9D, 0x38, 0xF5, 0xBC, 0xB6, 0xDA, 0x21, 0x10, 0xFF, 0xF3, 0xD2,  
0xCD, 0x0C, 0x13, 0xEC, 0x5F, 0x97, 0x44, 0x17, 0xC4, 0xA7, 0x7E, 0x3D, 0x64, 0x5D, 0x19, 0x73,  
0x60, 0x81, 0x4F, 0xDC, 0x22, 0x2A, 0x90, 0x88, 0x46, 0xEE, 0xB8, 0x14, 0xDE, 0x5E, 0x0B, 0xDB,  
0xE0, 0x32, 0x3A, 0x0A, 0x49, 0x06, 0x24, 0x5C, 0xC2, 0xD3, 0xAC, 0x62, 0x91, 0x95, 0xE4, 0x79,  
0xE7, 0xC8, 0x37, 0x6D, 0x8D, 0xD5, 0x4E, 0xA9, 0x6C, 0x56, 0xF4, 0xEA, 0x65, 0x7A, 0xAE, 0x08,  
0xBA, 0x78, 0x25, 0x2E, 0x1C, 0xA6, 0xB4, 0xC6, 0xE8, 0xDD, 0x74, 0x1F, 0x4B, 0xBD, 0x8B, 0x8A,  
0x70, 0x3E, 0xB5, 0x66, 0x48, 0x03, 0xF6, 0x0E, 0x61, 0x35, 0x57, 0xB9, 0x86, 0xC1, 0x1D, 0x9E,
```

```
0xE1, 0xF8, 0x98, 0x11, 0x69, 0xD9, 0x8E, 0x94, 0x9B, 0x1E, 0x87, 0xE9, 0xCE, 0x55, 0x28, 0xDF,
0x8C, 0xA1, 0x89, 0x0D, 0xBF, 0xE6, 0x42, 0x68, 0x41, 0x99, 0x2D, 0x0F, 0xB0, 0x54, 0xBB, 0x16 }
```

S-Box inverz

```
sbox_inverz[256] = {
0x52, 0x09, 0x6A, 0xD5, 0x30, 0x36, 0xA5, 0x38, 0xBF, 0x40, 0xA3, 0x9E, 0x81, 0xF3, 0xD7, 0xFB,
0x7C, 0xE3, 0x39, 0x82, 0x9B, 0x2F, 0xFF, 0x87, 0x34, 0x8E, 0x43, 0x44, 0xC4, 0xDE, 0xE9, 0xCB,
0x54, 0x7B, 0x94, 0x32, 0xA6, 0xC2, 0x23, 0x3D, 0xEE, 0x4C, 0x95, 0x0B, 0x42, 0xFA, 0xC3, 0x4E,
0x08, 0x2E, 0xA1, 0x66, 0x28, 0xD9, 0x24, 0xB2, 0x76, 0x5B, 0xA2, 0x49, 0x6D, 0x8B, 0xD1, 0x25,
0x72, 0xF8, 0xF6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xD4, 0xA4, 0x5C, 0xCC, 0x5D, 0x65, 0xB6, 0x92,
0x6C, 0x70, 0x48, 0x50, 0xFD, 0xED, 0xB9, 0xDA, 0x5E, 0x15, 0x46, 0x57, 0xA7, 0x8D, 0x9D, 0x84,
0x90, 0xD8, 0xAB, 0x00, 0x8C, 0xBC, 0xD3, 0x0A, 0xF7, 0xE4, 0x58, 0x05, 0xB8, 0xB3, 0x45, 0x06,
0xD0, 0x2C, 0x1E, 0x8F, 0xCA, 0x3F, 0x0F, 0x02, 0xC1, 0xAF, 0xBD, 0x03, 0x01, 0x13, 0x8A, 0x6B,
0x3A, 0x91, 0x11, 0x41, 0x4F, 0x67, 0xDC, 0xEA, 0x97, 0xF2, 0xCF, 0xCE, 0xF0, 0xB4, 0xE6, 0x73,
0x96, 0xAC, 0x74, 0x22, 0xE7, 0xAD, 0x35, 0x85, 0xE2, 0xF9, 0x37, 0xE8, 0x1C, 0x75, 0xDF, 0x6E,
0x47, 0xF1, 0x1A, 0x71, 0x1D, 0x29, 0xC5, 0x89, 0x6F, 0xB7, 0x62, 0x0E, 0xAA, 0x18, 0xBE, 0x1B,
0xFC, 0x56, 0x3E, 0x4B, 0xC6, 0xD2, 0x79, 0x20, 0x9A, 0xDB, 0xC0, 0xFE, 0x78, 0xCD, 0x5A, 0xF4,
0x1F, 0xDD, 0xA8, 0x33, 0x88, 0x07, 0xC7, 0x31, 0xB1, 0x12, 0x10, 0x59, 0x27, 0x80, 0xEC, 0x5F,
0x60, 0x51, 0x7F, 0xA9, 0x19, 0xB5, 0x4A, 0x0D, 0x2D, 0xE5, 0x7A, 0x9F, 0x93, 0xC9, 0x9C, 0xEF,
0xA0, 0xE0, 0x3B, 0x4D, 0xAE, 0x2A, 0xF5, 0xB0, 0xC8, 0xEB, 0xBB, 0x3C, 0x83, 0x53, 0x99, 0x61,
0x17, 0x2B, 0x04, 0x7E, 0xBA, 0x77, 0xD6, 0x26, 0xE1, 0x69, 0x14, 0x63, 0x55, 0x21, 0x0C, 0x7D }
```

Dodatak B: Ispitni vektori

128-bitni ključ

Ključ (hex): 2b7e151628aed2a6abf7158809cf4f3c

Jasni tekst (hex): 6bc1bee22e409f96e93d7e117393172a

Izlaz (hex): 3ad77bb40d7a3660a89ecaf32466ef97

Ispis po koracima:

0._ulaz	6bc1bee22e409f96e93d7e117393172a
0.kljuc	2b7e151628aed2a6abf7158809cf4f3c
0.d_klj	40bfabf406ee4d3042ca6b997a5c5816
1._ulaz	40bfabf406ee4d3042ca6b997a5c5816
1.z_okt	090862bf6f28e3042c747feeda4a6a47
1.p_red	09287f476f746abf2c4a6204da08e3ee
1.p_stu	529f16c2978615cae01aae54bala2659
1.kljuc	a0fafe1788542cb123a339392a6c7605
1.d_klj	f265e8d51fd2397bc3b9976d9076505c
2._ulaz	f265e8d51fd2397bc3b9976d9076505c
2.z_okt	894d9b03c0b512212e56883c6038534a

2.p_red	89b5884ac05653032e389b21604d123c
2.p_stu	0f31e929319a3558aec9589339f04d87
2.kljuc	f2c295f27a96b9435935807a7359f67f
2.d_klj	fdf37cdb4b0c8c1bf7fcd8e94aa9bbf8
3._ulaz	fdf37cdb4b0c8c1bf7fcd8e94aa9bbf8
3.z_okt	540d10b9b3fe64af68b0611ed6d3ea41
3.p_red	54fe6141b3b0eab968d310afd60d641e
3.p_stu	9151abe1e5541cfd014a713eda7e3134
3.kljuc	3d80477d4716fe3e1e237e446d7a883b
3.d_klj	acd1ec9ca242e2c31f690f7ab704b90f
4._ulaz	acd1ec9ca242e2c31f690f7ab704b90f
4.z_okt	913ecede3a2c982ec0f976daa9f25676
4.p_red	912c76763af956dec0f2ce2ea93e98da
4.p_stu	4d25cb1eecf716467658c73b49bcc9e9
4.kljuc	ef44a541a8525b7fb671253bdb0bad00
4.d_klj	a2616e5f44a54d39c029e20092b764e9
5._ulaz	a2616e5f44a54d39c029e20092b764e9
5.z_okt	3aef9fcf1b06e312baa598634fa9431e
5.p_red	3a06981e1ba543cfbaa99f124fefe363
5.p_stu	f89b35ec4e40724e025b00c734d7d81b
5.kljuc	d4d1c6f87c839d87caf2b8bc11f915bc
5.d_klj	2c4af31432c3efc9c8a9b87b252ecda7
6._ulaz	2c4af31432c3efc9c8a9b87b252ecda7
6.z_okt	71d60dfa232edfdde8d36c213f31bd5c
6.p_red	712e6c5c23d3bdfae8310ddd3fd6df21
6.p_stu	a0c563696fb884e44840bfbbee1d32f0a
6.kljuc	6d88a37a110b3efddbfb98641ca0093fd
6.d_klj	cd4dc0137eb3ba1993b939ff2bd3bcf7
7._ulaz	cd4dc0137eb3ba1993b939ff2bd3bcf7
7.z_okt	bde3ba7df36df4d4dc561216f1666568
7.p_red	bd6d1268f356657ddc66bad4f1e3f416
7.p_stu	ac394c731f8de8c76711b210253ddb33
7.kljuc	4e54f70e5f5fc9f384a64fb24ea6dc4f
7.d_klj	e26dbb7d40d22134e3b7fda26b9b077c
8._ulaz	e26dbb7d40d22134e3b7fda26b9b077c
8.z_okt	983ceaff09b5fd1811a9543a7f14c510
8.p_red	98b5541009a9c5ff1114ea187f3cfd3a
8.p_stu	ab05b572c8eb2b92ec04e2fd7d21ec34
8.kljuc	ead27321b58dbad2312bf5607f8d292f
8.d_klj	41d7c6537d669140dd2f179d02acc51b
9._ulaz	41d7c6537d669140dd2f179d02acc51b
9.z_okt	830eb4edff338109c115f05e7791a6af
9.p_red	8333f0afff15a6edc191b409770e815e
9.p_stu	1741a11891c991688c36386f23ad82aa
9.kljuc	ac7766f319fadc2128d12941575c006e
9.d_klj	bb36c7eb88334d49a4e7112e74f182c4
10._ulaz	bb36c7eb88334d49a4e7112e74f182c4
10.z_okt	ea05c6e9c4c3e33b4994823192a1131c
10.p_red	eac3821cc49413e949a1c63b9205e331
10.kljuc	d014f9a8c9ee2589e13f0cc8b6630ca6
10.d_klj	3ad77bb40d7a3660a89ecaf32466ef97

256-bitni ključ

Ključ (hex): 603deb1015ca71be2b73aef0857d77811f352c073b6108d72d9810a30914dff4

Jasni tekst (hex): 6bc1bee22e409f96e93d7e117393172a

Izlaz (hex): f3eed1bdb5d2a03c064b5a7e3db181f8

Ispis po koracima:

0._ulaz	6bc1bee22e409f96e93d7e117393172a
0.kljuc	603deb1015ca71be2b73aef0857d7781
0.d_klj	0bfc55f23b8aee28c24ed0e1f6ee60ab
1._ulaz	0bfc55f23b8aee28c24ed0e1f6ee60ab
1.z_okt	2bb0fc89e27e2834252f70f84228d062
1.p_red	2b7e7062e22fd0892528fc3442b028f8
1.p_stu	c62513b7f75ef6cbfa5eb2d39fb9b1b5
1.kljuc	1f352c073b6108d72d9810a30914dff4
1.d_klj	d9103fb0cc3ffe1cd7c6a27096ad6e41
2._ulaz	d9103fb0cc3ffe1cd7c6a27096ad6e41
2.z_okt	35ca75e74b75bb9c0eb43a5190959f83
2.p_red	35753a834bb49fe70e95759c90cabb51
2.p_stu	4c12aa0d2965e823513cced19498c478
2.kljuc	9ba354118e6925afa51a8b5f2067fcde
2.d_klj	d7b1felca70ccd8cf426458eb4ff38a6
3._ulaz	d7b1felca70ccd8cf426458eb4ff38a6
3.z_okt	0ec8bb9c5cfebd64bff76e198d160724
3.p_red	0efe6e245cf7079cbf16bb648dc8bd19
3.p_stu	4f7f40ca213c1a37802168bfe6c30fcb
3.kljuc	a8b09c1a93d194cdbe49846eb75d5b9a
3.d_klj	e7cfdcd0b2ed8efa3e68ecd1519e5451
4._ulaz	e7cfdcd0b2ed8efa3e68ecd1519e5451
4.z_okt	948a86703755192db245ce3ed10b20d1
4.p_red	9455ced137452070b20b862dd18a193e
4.p_stu	d3a62e85f1ada2dcc918d91a1bcb2b87
4.kljuc	d59aecb85bf3c917fee94248de8ebe96
4.d_klj	063cc23daa5e6bcb37f19b52c5459511
5._ulaz	063cc23daa5e6bcb37f19b52c5459511
5.z_okt	6feb2527ac587f1f9aa11400a66e2a82
5.p_red	6f581482aca12a279a6e251fa6eb7f00
5.p_stu	a06182e2b6ac302aa7369fc00eeab365
5.kljuc	b5a9328a2678a647983122292f6c79b3
5.d_klj	15c8b06890d4966d3f07bde92186cad6
6._ulaz	15c8b06890d4966d3f07bde92186cad6
6.z_okt	59e8e7456048903c75c57a1efd4474f6
6.p_red	59487af660c574457544e73cfde8901e
6.p_stu	e6b1e42ea528829bfdf3a0444c830c58
6.kljuc	812c81addadf48ba24360af2fab8b464
6.d_klj	679d65837ff7ca21d9c5aab6b63bb83c
7._ulaz	679d65837ff7ca21d9c5aab6b63bb83c

7.z_okt	855e4decd26874fd35a6ac4e4ee26ceb
7.p_red	8568acebd2a66cec35e24dfd4e5e744e
7.p_stu	ee51889dcedd8364e7c0511144202a64
7.kljuc	98c5bfc9bebd198e268c3ba709e04214
7.d_klj	7694375470609aeac14c6ab64dc06870
8._ulaz	7694375470609aeac14c6ab64dc06870
8.z_okt	38229a2051d0b8877829024ee3ba4551
8.p_red	38d002515129452078ba9a87e322b84e
8.p_stu	48d41f38bcec92df38257fbd4d3a7838
8.kljuc	68007bacb2df331696e939e46c518d80
8.d_klj	20d464940e33a1c9aecc4659216bf5b8
9._ulaz	20d464940e33a1c9aecc4659216bf5b8
9.z_okt	b7484322abc332dde44b5acb7d7fe66c
9.p_red	b7c35a6cab4be622e47f43ddfd4832cb
9.p_stu	1da87483542e510fcc0261aac0f097eb
9.kljuc	c814e20476a9fb8a5025c02d59c58239
9.d_klj	d5bc96872287aa859c27a187993515d2
10._ulaz	d5bc96872287aa859c27a187993515d2
10.z_okt	036590179317ac97decc3217ee9659b5
10.p_red	031732b593cc5917de969097ee65ac17
10.p_stu	b8ceb4513cecd41501d5d14ad3dcf1ce
10.kljuc	de1369676ccc5a71fa2563959674ee15
10.d_klj	66ddddd3650208e64fbf0b2df45a81fdb
11._ulaz	66ddddd3650208e64fbf0b2df45a81fdb
11.z_okt	33c1c10553b719430f8c379e6ec2c0b9
11.p_red	33b737b9538cc0050fc2c1436ec1199e
11.p_stu	2aa63abcec0e4bb3c18b91940342244d
11.kljuc	5886ca5d2e2f31d77e0af1fa27cf73c3
11.d_klj	7220f0e1c2217a64bf81606e248d578e
12._ulaz	7220f0e1c2217a64bf81606e248d578e
12.z_okt	40b78cf825fdda43080cd09f365d5b19
12.p_red	40fdd019250c5bf8085d8c4336b7da9f
12.p_stu	55d32ddffd288cd3387e934feba99412
12.kljuc	749c47ab18501ddae2757e4f7401905a
12.d_klj	214f6a74e5789109da0bed009fa80448
13._ulaz	214f6a74e5789109da0bed009fa80448
13.z_okt	fd840292d9bc8101572b5563dbc2f252
13.p_red	fdbc5552d92bf29257c20201db848163
13.p_stu	39331d51b410a096f0cf923bd833e3b5
13.kljuc	cafaaae3e4d59b349adf6acebd10190d
13.d_klj	f3c9b7b250c53ba26a10f8f56523fab8
14._ulaz	f3c9b7b250c53ba26a10f8f56523fab8
14.z_okt	0ddda93753a6e23a02ca41e64d262d6c
14.p_red	0da6416c53ca2d370226a93a4ddde2e6
14.kljuc	fe4890d1e6188d0b046df344706c631e
14.d_klj	f3eed1bdb5d2a03c064b5a7e3db181f8

Literatura

1. Operacijski sustavi; Budin, L.; Golub, M; Jakobovic, D., Jelenkovic, L; Element, Zagreb; 2010

2. A Specification for Rijndael, the AES Algorithm, Dr. Brian Gladman, v3.11, 12th Sept 2003
3. http://en.wikipedia.org/wiki/Advanced_Encryption_Standard