UNIVERSITETI I PRISHTINES FAKULTETI I SHKENCAVE MATEMATIKO-NATYRORE

Tema: AES i thjeshtuar me CTR

Lënda: Siguri e të Dhënave

Punuar nga:

Ardit Maliqi
Atlant Klaiqi
Andi Alidema

Ligjëruesit e lëndës:

Prof. Artan Berisha
Asst.Besnik Duriqi

Mënyra e funksionimit të AES dhe përdorimi tij

Sigurimi i të dhënave është një aspekt thelbësor në përdorimin e internetit, dhe AES (Advanced Encryption Standard) është një algoritëm simetrik i kriptimit i përdorur gjerësisht për të mbrojtur të dhënat sensitive. AES punon duke marrë një bllok plaintext dhe duke e kriptuar atë në një bllok ciphertext duke përdorur një çelës simetrik. Ky çelës përdoret për të skajtur plaintext në një format të panjohur, dhe ciphertext-i i rezultuar mund të dekriptohet vetëm me të njëjtin çelës.

Procesi i kriptimit fillon duke ndarë bllokun plaintext në pjesë më të vogla prej 128 bitësh. Çdo pjesë pastaj kalon nëpër një seri operacionesh matematikore, duke përfshirë zëvendësim, permutim dhe bitëise XOR. Këto operacione janë projektuar për të skajtur të dhënat në një mënyrë që bëjnë të pamundur leximin e tyre pa përdorur çelësin e duhur. Ciphertext-i i rezultuar pastaj transmetohet ose ruhet me siguri.

Një nga avantazhet kryesore të AES është aftësia e tij për të përdorur një gamë madhësie çelësish, duke përfshirë 128, 192 dhe 256 bitë. Sa më e madhe madhësia e çelësit, aq më e sigurt është kriptimi, pasi ka më shumë kombinime të mundshme të çelësave që mund të përdoren për të kriptuar dhe dekriptuar të dhënat. AES përdor një proces të quajtur zgjerimi i çelësit për të gjeneruar nënçelësat që përdoren në procesin e kriptimit.

AES përdoret gjerësisht në një gamë të gjerë aplikacionesh ku siguria e të dhënave është thelbësore, duke përfshirë sigurimin e të dhënave në tranzit në internet, kriptimin e skedarëve të ruajtur në hard disk dhe mbrojtjen e informacionit të ndjeshëm si numrat e kartave të kreditit dhe fjalëkalimet. Gjithashtu përdoret nga qeveritë dhe organizatat ushtarake për të mbrojtur informacionin klasifikuar. AES konsiderohet si shumë i sigurt dhe është bërë standardi de facto për kriptimin simetrik.

Përveç përfitimeve të sigurisë së tij, AES është gjithashtu shumë shpejt dhe efikas, duke e bërë atë një zgjedhje ideale për përdorim në sistemet e kohës reale dhe aplikacionet e internetit që kërkojnë një procesim të shpejtë të të dhënave. Në përgjithësi, AES është një teknologji që mund të sigurojë një nivel të lartë të sigurisë për të dhënat e ndjeshme, dhe është shumë e rëndësishme për të mbrojtur të dhënat tona në një botë ku kërcënimet e sigurisë janë të pranishme në mënyra të ndryshme.

Megjithëse AES është shumë i sigurt, nuk ka asnjë sistem kriptimi që është plotësisht i pandreqshëm. Sistemet e kriptimit, përfshirë AES, mund të thyhen nga sulmet e ndryshme të kriptografisë, përfshirë sulmet bruteforce, sulmet e ndërmjetësimit dhe sulmet e injektimit të kodit. Për këtë arsye, është e rëndësishme që të gjithë që përdorin AES dhe teknologji të tjera të sigurisë të kenë një strategji të përgjithshme të sigurisë që përfshin mbajtjen e çelësave të sigurisë dhe përdorimin e protokolleve të sigurisë së të dhënave të sigurta.

Mënyra e funksionimit të Counter Mode(CTR) dhe përdorimi tij

Counter Mode (CTR) është një mënyrë e veprimit për shifrat e blloqeve, si AES (Advanced Encryption Standard), që përdoret gjerësisht në sigurinë e të dhënave për të siguruar konfidencialitetin dhe integritetin e informacionit të ndjeshëm.

CTR mode është një shifër rrjedhëse që lejon një shifër blloku të krijojë blloqe të ndryshme të të dhënave të pavarura. Ajo vepron duke prodhuar një rrjedhë unike të çelësave për çdo bllok të tekstit të pastër, që pastaj kombinohet me tekstin e pastër duke përdorur një operacion XOR për të prodhuar tekstin e shifrave. Rrjedha e çelësave prodhohet duke kodifikuar një vlerë të numratorit duke përdorur shifrën e bllokut dhe tekstin e shifrave të prodhuara përdoren si rrjedhë e çelësave për bllokun e përafërt të tekstit të pastër. Vlera e numratorit rritet për çdo bllok i mëtejshëm, duke siguruar që rrjedha e çelësave është unike për çdo bllok.

Një nga përfitimet kryesore të CTR mode është aftësia e saj për të kodifikuar sasi të mëdha të të dhënave në një model akses i rastësishëm. Kjo do të thotë se çdo bllok i të dhënave mund të kodifikohet dhe dekodifikohet pavarësisht nga pjesa tjetër e të dhënave. Kjo është veçanërisht e dobishme në situata kur duhet të kodifikohen ose dekodifikohen skedarë të mëdhenj, pasi që lejon procesimin e efektshëm të të dhënave.

Një përfitim tjetër i CTR mode është rezistenca e saj ndaj disa llojeve të sulmeve, si sulmet në tekstin e shifrave. Kjo për shkak se rrjedha e çelësave nuk vjen nga teksti i pastër, duke bërë të vështirë për një sulmues të modifikojë tekstin e shifrave pa u zbuluar.

Për të përdorur CTR mode në sigurinë e të dhënave, një organizatë zakonisht do të integrojë në protokollet e saj ekzistuese të kodifikimit dhe do të sigurojë që të gjitha të dhënat e ndjeshme të kodifikohen duke përdorur këtë mënyrë.

Historia e AES të thjeshtësuar me CTR

AES të thjeshtësuar me CTR (Counter) është një variant i Advanced Encryption Standard (AES) që përdor një version të thjeshtësuar të algoritmit AES me një mënyrë veprimi të numrit të llogaritësit (CTR).

Algoritmi AES u zhvillua nga Joan Daemen dhe Vincent Rijmen dhe u miratua nga qeveria amerikane si një standard për kodifikim në vitin 2001. AES është një shifra e bllokut, që do të thotë se vepron në blloqe të dhënash me gjatësi të fiksuar, dhe është gjerësisht përdorur për të siguruar të dhënat në shumë aplikacione.

AES thjeshtësuar me CTR u zhvillua si një mënyrë për të ofruar një implementim më të thjeshtë dhe më të lehtë në peshë të AES që mund të përdoret në mjediset me kufizime burimesh, si në pajisjet e Internetit të Gjërave (IoT) ose në sistemet e integruara.

Në AES të thjeshtësuar, algoritmi AES origjinal është thjeshtësuar duke reduktuar numrin e raundeve dhe madhësinë e çelësit dhe bllokut. Algoritmi i thjeshtësuar përdor një bllok prej

128-bit dhe një çelës prej 128-bit, dhe kryen vetëm dy raunde kodifikimi në vend të 10 ose 14 raundeve të zakonshme të përdorura në AES.

Përveç thjeshtësimeve në algoritmin AES, AES i thjeshtësuar me CTR gjithashtu përdor modën e veprimtarisë së numrit të llogaritësit (CTR), që është një mënyrë e kodifikimit të të dhënave në të cilën një numër llogaritësi përdoret për të gjeneruar një rrjedhë numrash pseudo-random, që pastaj janë XOR me tekst të qartë për të prodhuar tekst të kodifikuar.

Përdorimi i modës CTR në AES i thjeshtësuar me CTR ofron disa avantazhe në krahasim me modet e tjera të veprimtarisë. Për shembull, lejon kodifikimin dhe dekodifikimin paralel, që mund të jetë i dobishëm në disa aplikacione. Gjithashtu ofron vetitë e sigurisë të mira, si rezistencën ndaj sulmeve të njohura të tekstit të qartë.

Në përgjithësi, AES i thjeshtësuar me CTR ofron një mënyrë të lehtë dhe efikase për të implementuar kodifikimin AES në mjediset me kufizime burimesh, duke siguruar njëkohësisht vetitë e sigurisë të forta.

Avantazhet dhe Disavantazhet e AES të thjeshtësuar me CTR

AES i thjeshtësuar me CTR (Counter) mode është një version i lehtësuar i Advanced Encryption Standard (AES) që përdor një madhësi më të vogël të çelësit dhe një algoritëm më të thjeshtë të kodifikimit. Këtu janë disa avantazhe dhe dezavantazhe të përdorimit të AES i thjeshtësuar me CTR:

Avantazhet:

- ➢ Kodifikimi dhe dekodifikimi më i shpejtë: AES i thjeshtësuar me CTR përdor një madhësi më të vogël të çelësit dhe një algoritëm më të thjeshtë të kodifikimit, që e bën më të shpejtë se algoritmi i AES origjinal. Kjo e bën atë një zgjedhje të mirë për sisteme që kërkojnë kodifikim dhe dekodifikim të shpejtë.
- ➢ Kërkesa më e ulët për memorie: Për shkak se AES i thjeshtësuar me CTR përdor një madhësi më të vogël të çelësit dhe një algoritëm më të thjeshtë, kërkon më pak memorie se algoritmi i AES origjinal. Kjo e bën atë një zgjedhje të mirë për sisteme me resurse të kufizuara të memorjes.
- ➢ Implementimi i thjeshtësuar: AES i thjeshtësuar me CTR është më e lehtë për t'u implementuar se algoritmi i AES origjinal, duke e bërë atë një zgjedhje të mirë për sisteme me fuqi të kufizuar të përpunimit ose kohë zhvillimi.

Disavantazhet:

- ➢ Siguria më e dobët: AES i thjeshtësuar me CTR përdor një madhësi më të vogël të çelësit dhe një algoritëm më të thjeshtë, që e bën atë më pak të sigurt se algoritmi i AES origjinal. Kjo e bën atë një zgjedhje të dobët për sisteme që kërkojnë nivele të larta të sigurisë.
- ➢ I ndjeshëm ndaj sulmeve: AES i thjeshtësuar me CTR është i ndjeshëm ndaj sulmeve si sulmet brutale dhe ato diferenciale, që mund të kompromentojnë sigurinë e sistemit.

- Madhësia e kufizuar e çelës: AES i thjeshtësuar me CTR ka një gamë të kufizuar të madhësive të çelësit, që mund të bëjë të vështirë përdorimin e tij në sisteme që kërkojnë madhësi të mëdha të çelësit për siguri më të mirë.

Kodi ne java
Kodi jonë përmban 5 klasa te rëndësishme:

Klasa Util.java dhe përmban metoda esenciale që duhet për të zhvilluar AES, kodi është si në vijim:

```
package cis.lib;

public class Util {


    public static byte[] short2byte (short[] sa) {
        int length = sa.length;
        byte[] ba = new byte[length * 2];
        for (int i = 0, j = 0, k; i < length; ) {
            k = sa[i++];
            ba[j++] = (byte)((k >>> 8) & 0xFF);
            ba[j++] = (byte)( k        & 0xFF);
        }
        return (ba);
    }

    public static short[] byte2short (byte[] ba) {
        int length = ba.length;
        short[] sa = new short[length / 2];
        for (int i = 0, j = 0; j < length / 2; ) {
            sa[j++] = (short)(((ba[i++] & 0xFF) <<  8) |
                    ((ba[i++] & 0xFF)      ));
        }
        return (sa);
    }

    public static byte[] int2byte (int[] ia) {
        int length = ia.length;
        byte[] ba = new byte[length * 4];
        for (int i = 0, j = 0, k; i < length; ) {
```

```
            k = ia[i++];
            ba[j++] = (byte)((k >>>24) & 0xFF);
            ba[j++] = (byte)((k >>>16) & 0xFF);
            ba[j++] = (byte)((k >>> 8) & 0xFF);
            ba[j++] = (byte)( k        & 0xFF);
        }
        return (ba);
    }

    public static int[] byte2int (byte[] ba) {
        int length = ba.length;
        int[] ia = new int[length / 4];
        for (int i = 0, j = 0; j < length / 4; ) {
            ia[j++] = (((ba[i++] & 0xFF) << 24) |
                    ((ba[i++] & 0xFF) << 16) |
                    ((ba[i++] & 0xFF) <<  8) |
                    ((ba[i++] & 0xFF)      ));
        }
        return (ia);
    }

    public static final char[] HEX_DIGITS = {

'0','1','2','3','4','5','6','7','8','9','a','b','c','d','e','f'
    };


    public static String toHEX (byte[] ba) {
        int length = ba.length;
        char[] buf = new char[length * 3];
        for (int i = 0, j = 0, k; i < length; ) {
            k = ba[i++];
            buf[j++] = HEX_DIGITS[(k >>> 4) & 0x0F];
            buf[j++] = HEX_DIGITS[ k        & 0x0F];
            buf[j++] = ' ';
        }
        return new String(buf);
    }

    public static String toHEX (short[] ia) {
        int length = ia.length;
        char[] buf = new char[length * 5];
        for (int i = 0, j = 0, k; i < length; ) {
```

```
            k = ia[i++];
            buf[j++] = HEX_DIGITS[(k >>>12) & 0x0F];
            buf[j++] = HEX_DIGITS[(k >>> 8) & 0x0F];
            buf[j++] = HEX_DIGITS[(k >>> 4) & 0x0F];
            buf[j++] = HEX_DIGITS[ k        & 0x0F];
            buf[j++] = ' ';
        }
        return new String(buf);
    }

    public static String toHEX (int[] ia) {
        int length = ia.length;
        char[] buf = new char[length * 10];
        for (int i = 0, j = 0, k; i < length; ) {
            k = ia[i++];
            buf[j++] = HEX_DIGITS[(k >>>28) & 0x0F];
            buf[j++] = HEX_DIGITS[(k >>>24) & 0x0F];
            buf[j++] = HEX_DIGITS[(k >>>20) & 0x0F];
            buf[j++] = HEX_DIGITS[(k >>>16) & 0x0F];
            buf[j++] = ' ';
            buf[j++] = HEX_DIGITS[(k >>>12) & 0x0F];
            buf[j++] = HEX_DIGITS[(k >>> 8) & 0x0F];
            buf[j++] = HEX_DIGITS[(k >>> 4) & 0x0F];
            buf[j++] = HEX_DIGITS[ k        & 0x0F];
            buf[j++] = ' ';
        }
        return new String(buf);
    }


    public static String toHEX1 (byte b) {
        char[] buf = new char[2];
        int j = 0;
        buf[j++] = HEX_DIGITS[(b >>> 4) & 0x0F];
        buf[j++] = HEX_DIGITS[ b        & 0x0F];
        return new String(buf);
    }

    public static String toHEX1 (byte[] ba) {
        int length = ba.length;
        char[] buf = new char[length * 2];
        for (int i = 0, j = 0, k; i < length; ) {
            k = ba[i++];
```

```
            buf[j++] = HEX_DIGITS[(k >>> 4) & 0x0F];
            buf[j++] = HEX_DIGITS[ k        & 0x0F];
        }
        return new String(buf);
    }


    public static String toHEX1 (short[] ia) {
        int length = ia.length;
        char[] buf = new char[length * 4];
        for (int i = 0, j = 0, k; i < length; ) {
            k = ia[i++];
            buf[j++] = HEX_DIGITS[(k >>>12) & 0x0F];
            buf[j++] = HEX_DIGITS[(k >>> 8) & 0x0F];
            buf[j++] = HEX_DIGITS[(k >>> 4) & 0x0F];
            buf[j++] = HEX_DIGITS[ k        & 0x0F];
        }
        return new String(buf);
    }


    public static String toHEX1 (int i) {
        char[] buf = new char[8];
        int j = 0;
        buf[j++] = HEX_DIGITS[(i >>>28) & 0x0F];
        buf[j++] = HEX_DIGITS[(i >>>24) & 0x0F];
        buf[j++] = HEX_DIGITS[(i >>>20) & 0x0F];
        buf[j++] = HEX_DIGITS[(i >>>16) & 0x0F];
        buf[j++] = HEX_DIGITS[(i >>>12) & 0x0F];
        buf[j++] = HEX_DIGITS[(i >>> 8) & 0x0F];
        buf[j++] = HEX_DIGITS[(i >>> 4) & 0x0F];
        buf[j++] = HEX_DIGITS[ i        & 0x0F];
        return new String(buf);
    }


    public static String toHEX1 (int[] ia) {
        int length = ia.length;
        char[] buf = new char[length * 8];
        for (int i = 0, j = 0, k; i < length; ) {
            k = ia[i++];
            buf[j++] = HEX_DIGITS[(k >>>28) & 0x0F];
            buf[j++] = HEX_DIGITS[(k >>>24) & 0x0F];
```

```
            buf[j++] = HEX_DIGITS[(k >>>20) & 0x0F];
            buf[j++] = HEX_DIGITS[(k >>>16) & 0x0F];
            buf[j++] = HEX_DIGITS[(k >>>12) & 0x0F];
            buf[j++] = HEX_DIGITS[(k >>> 8) & 0x0F];
            buf[j++] = HEX_DIGITS[(k >>> 4) & 0x0F];
            buf[j++] = HEX_DIGITS[ k        & 0x0F];
        }
        return new String(buf);
    }


    public static byte[] hex2byte(String hex) {
        int len = hex.length();
        byte[] buf = new byte[((len + 1) / 2)];

        int i = 0, j = 0;
        if ((len % 2) == 1)
            buf[j++] = (byte) hexDigit(hex.charAt(i++));

        ëhile (i < len) {
            buf[j++] = (byte) ((hexDigit(hex.charAt(i++)) << 4) |
                    hexDigit(hex.charAt(i++)));
        }
        return buf;
    }


    public static boolean isHex(String hex) {
        int len = hex.length();
        int i = 0;
        char ch;

        ëhile (i < len) {
            ch = hex.charAt(i++);
            if (! ((ch >= '0' && ch <= '9') || (ch >= 'A' && ch <= 'F')
||
                    (ch >= 'a' && ch <= 'f'))) return false;
        }
        return true;
    }

    public static int hexDigit(char ch) {
        if (ch >= '0' && ch <= '9')
```

```
            return ch - '0';
        if (ch >= 'A' && ch <= 'F')
            return ch - 'A' + 10;
        if (ch >= 'a' && ch <= 'f')
            return ch - 'a' + 10;

        return(0);
    }

}
```

Klasa AES bën implementimin e Advancdes Ecryption Standart dhe kodi është si në vijim:

```
package cis.lib;

import java.util.*;


public class AES {


    public int traceLevel = 0;


    public String traceInfo = "";

    public static final int
            ROUNDS = 14,
            BLOCK_SIZE = 16,
            KEY_LENGTH = 32;


    int numRounds;

    byte[][] Ke;

    byte[][] Kd;


    static final byte[] S = {
            99, 124, 119, 123, -14, 107, 111, -59, 48, 1, 103, 43, -2,
-41, -85, 118,
```

```
            -54, -126, -55, 125, -6, 89, 71, -16, -83, -44, -94, -81,
-100, -92, 114, -64,
            -73, -3, -109, 38, 54, 63, -9, -52, 52, -91, -27, -15, 113,
-40, 49, 21,
            4, -57, 35, -61, 24, -106, 5, -102, 7, 18, -128, -30, -21,
39, -78, 117,
            9, -125, 44, 26, 27, 110, 90, -96, 82, 59, -42, -77, 41,
-29, 47, -124,
            83, -47, 0, -19, 32, -4, -79, 91, 106, -53, -66, 57, 74,
76, 88, -49,
            -48, -17, -86, -5, 67, 77, 51, -123, 69, -7, 2, 127, 80,
60, -97, -88,
            81, -93, 64, -113, -110, -99, 56, -11, -68, -74, -38, 33,
16, -1, -13, -46,
            -51, 12, 19, -20, 95, -105, 68, 23, -60, -89, 126, 61, 100,
93, 25, 115,
            96, -127, 79, -36, 34, 42, -112, -120, 70, -18, -72, 20,
-34, 94, 11, -37,
            -32, 50, 58, 10, 73, 6, 36, 92, -62, -45, -84, 98, -111,
-107, -28, 121,
            -25, -56, 55, 109, -115, -43, 78, -87, 108, 86, -12, -22,
101, 122, -82, 8,
            -70, 120, 37, 46, 28, -90, -76, -58, -24, -35, 116, 31, 75,
-67, -117, -118,
            112, 62, -75, 102, 72, 3, -10, 14, 97, 53, 87, -71, -122,
-63, 29, -98,
            -31, -8, -104, 17, 105, -39, -114, -108, -101, 30, -121,
-23, -50, 85, 40, -33,
            -116, -95, -119, 13, -65, -26, 66, 104, 65, -103, 45, 15,
-80, 84, -69, 22 };


    static final byte[] Si = {
            82, 9, 106, -43, 48, 54, -91, 56, -65, 64, -93, -98, -127,
-13, -41, -5,
            124, -29, 57, -126, -101, 47, -1, -121, 52, -114, 67, 68,
-60, -34, -23, -53,
            84, 123, -108, 50, -90, -62, 35, 61, -18, 76, -107, 11, 66,
-6, -61, 78,
            8, 46, -95, 102, 40, -39, 36, -78, 118, 91, -94, 73, 109,
-117, -47, 37,
            114, -8, -10, 100, -122, 104, -104, 22, -44, -92, 92, -52,
93, 101, -74, -110,
```

```
            108, 112, 72, 80, -3, -19, -71, -38, 94, 21, 70, 87, -89,
-115, -99, -124,
            -112, -40, -85, 0, -116, -68, -45, 10, -9, -28, 88, 5, -72,
-77, 69, 6,
            -48, 44, 30, -113, -54, 63, 15, 2, -63, -81, -67, 3, 1, 19,
-118, 107,
            58, -111, 17, 65, 79, 103, -36, -22, -105, -14, -49, -50,
-16, -76, -26, 115,
            -106, -84, 116, 34, -25, -83, 53, -123, -30, -7, 55, -24,
28, 117, -33, 110,
            71, -15, 26, 113, 29, 41, -59, -119, 111, -73, 98, 14, -86,
24, -66, 27,
            -4, 86, 62, 75, -58, -46, 121, 32, -102, -37, -64, -2, 120,
-51, 90, -12,
            31, -35, -88, 51, -120, 7, -57, 49, -79, 18, 16, 89, 39,
-128, -20, 95,
            96, 81, 127, -87, 25, -75, 74, 13, 45, -27, 122, -97, -109,
-55, -100, -17,
            -96, -32, 59, 77, -82, 42, -11, -80, -56, -21, -69, 60,
-125, 83, -103, 97,
            23, 43, 4, 126, -70, 119, -42, 38, -31, 105, 20, 99, 85,
33, 12, 125 };


    static final byte[] rcon = {
            0,
            1, 2, 4, 8, 16, 32,
            64, -128, 27, 54, 108, -40,
            -85, 77, -102, 47, 94, -68,
            99, -58, -105, 53, 106, -44,
            -77, 125, -6, -17, -59, -111 };

    public static final int
            COL_SIZE = 4,
            NUM_COLS = BLOCK_SIZE / COL_SIZE,
            ROOT = 0x11B;


    static final int[] roë_shift = {0, 1, 2, 3};


    static final int[] alog = new int[256];
```

```
    static final int[] log =  new int[256];


    static {
        int i, j;

        alog[0] = 1;
        for (i = 1; i < 256; i++) {
            j = (alog[i-1] << 1) ^ alog[i-1];
            if ((j & 0x100) != 0) j ^= ROOT;
            alog[i] = j;
        }
        for (i = 1; i < 255; i++) log[alog[i]] = i;
    }


    public AES() {
    }

    public static int getRounds (int keySize) {
        sëitch (keySize) {
            case 16:
                return 10;
            case 24:
                return 12;
            default:
                return 14;
        }
    }

    static final int mul (int a, int b) {
        return (a != 0 && b != 0) ?
                alog[(log[a & 0xFF] + log[b & 0xFF]) % 255] :
                0;
    }


    public static void trace_static() {
        int i,j;
        System.out.print("AES Static Tables\n");
        System.out.print("S[] = \n"); for(i=0;i<16;i++) {
for(j=0;j<16;j++) System.out.print(Util.toHEX1(S[i*16+j])+", ");
System.out.println();}
```

```
        System.out.print("Si[] = \n"); for(i=0;i<16;i++) {
for(j=0;j<16;j++) System.out.print(Util.toHEX1(Si[i*16+j])+", ");
System.out.println();}
        System.out.print("rcon[] = \n"); for(i=0;i<5;i++)
{for(j=0;j<6;j++) System.out.print(Util.toHEX1(rcon[i*6+j])+", ");
System.out.println();}
        System.out.print("log[] = \n"); for(i=0;i<32;i++)
{for(j=0;j<8;j++) System.out.print(Util.toHEX1(log[i*8+j])+", ");
System.out.println();}
        System.out.print("alog[] = \n"); for(i=0;i<32;i++)
{for(j=0;j<8;j++) System.out.print(Util.toHEX1(alog[i*8+j])+", ");
System.out.println();}
    }

    public byte[] encrypt(byte[] plain) {

        byte [] a = new byte[BLOCK_SIZE];
        byte [] ta = new byte[BLOCK_SIZE];
        byte [] Ker;
        int i, j, k, roë, col;

        traceInfo = "";
        if (traceLevel > 0) traceInfo = "encryptAES(" +
Util.toHEX1(plain) + ")";


        if (plain == null)
            throë new IllegalArgumentException("Empty plaintext");
        if (plain.length != BLOCK_SIZE)
            throë new IllegalArgumentException("Incorrect plaintext
length");


        Ker = Ke[0];
        for (i = 0; i < BLOCK_SIZE; i++)    a[i] = (byte)(plain[i] ^
Ker[i]);
        if (traceLevel > 2)
            traceInfo += "\n  R0 (Key = "+Util.toHEX1(Ker)+")\n\tAK =
"+Util.toHEX1(a);
        else if (traceLevel > 1)
            traceInfo += "\n  R0 (Key = "+Util.toHEX1(Ker)+")\t =
"+Util.toHEX1(a);
```

```
        for (int r = 1; r < numRounds; r++) {
            Ker = Ke[r];
            if (traceLevel > 1) traceInfo += "\n  R"+r+" (Key =
"+Util.toHEX1(Ker)+")\t";


            for (i = 0; i < BLOCK_SIZE; i++) ta[i] = S[a[i] & 0xFF];
            if (traceLevel > 2) traceInfo += "\n\tSB =
"+Util.toHEX1(ta);


            for (i = 0; i < BLOCK_SIZE; i++) {
                roë = i % COL_SIZE;
                k = (i + (roë_shift[roë] * COL_SIZE)) % BLOCK_SIZE;
                a[i] = ta[k];
            }
            if (traceLevel > 2) traceInfo += "\n\tSR =
"+Util.toHEX1(a);


            for (col = 0; col < NUM_COLS; col++) {
                i = col * COL_SIZE;
                ta[i]   = (byte)(mul(2,a[i]) ^ mul(3,a[i+1]) ^ a[i+2] ^
a[i+3]);
                ta[i+1] = (byte)(a[i] ^ mul(2,a[i+1]) ^ mul(3,a[i+2]) ^
a[i+3]);
                ta[i+2] = (byte)(a[i] ^ a[i+1] ^ mul(2,a[i+2]) ^
mul(3,a[i+3]));
                ta[i+3] = (byte)(mul(3,a[i]) ^ a[i+1] ^ a[i+2] ^
mul(2,a[i+3]));
            }
            if (traceLevel > 2) traceInfo += "\n\tMC =
"+Util.toHEX1(ta);


            for (i = 0; i < BLOCK_SIZE; i++)    a[i] = (byte)(ta[i] ^
Ker[i]);
            if (traceLevel > 2) traceInfo += "\n\tAK";
            if (traceLevel > 1) traceInfo += " = "+Util.toHEX1(a);
        }
```

```
        Ker = Ke[numRounds];
        if (traceLevel > 1) traceInfo += "\n  R"+numRounds+" (Key =
"+Util.toHEX1(Ker)+")\t";


        for (i = 0; i < BLOCK_SIZE; i++) a[i] = S[a[i] & 0xFF];
        if (traceLevel > 2) traceInfo += "\n\tSB = "+Util.toHEX1(a);


        for (i = 0; i < BLOCK_SIZE; i++) {
            roë = i % COL_SIZE;
            k = (i + (roë_shift[roë] * COL_SIZE)) % BLOCK_SIZE;
            ta[i] = a[k];
        }
        if (traceLevel > 2) traceInfo += "\n\tSR = "+Util.toHEX1(a);


        for (i = 0; i < BLOCK_SIZE; i++)    a[i] = (byte)(ta[i] ^
Ker[i]);
        if (traceLevel > 2) traceInfo += "\n\tAK";
        if (traceLevel > 1) traceInfo += " = "+Util.toHEX1(a)+"\n";
        if (traceLevel > 0) traceInfo += " = "+Util.toHEX1(a)+"\n";
        return (a);
    }



    public byte[] decrypt(byte[] cipher) {

        byte [] a = new byte[BLOCK_SIZE];
        byte [] ta = new byte[BLOCK_SIZE];
        byte [] Kdr;
        int i, j, k, roë, col;

        traceInfo = "";
        if (traceLevel > 0) traceInfo = "decryptAES(" +
Util.toHEX1(cipher) + ")";


        if (cipher == null)
            throë new IllegalArgumentException("Empty ciphertext");
        if (cipher.length != BLOCK_SIZE)
```

```
            throë new IllegalArgumentException("Incorrect ciphertext
length");


        Kdr = Kd[0];
        for (i = 0; i < BLOCK_SIZE; i++)    a[i] = (byte)(cipher[i] ^
Kdr[i]);
        if (traceLevel > 2)
            traceInfo += "\n  R0 (Key = "+Util.toHEX1(Kdr)+")\n\t AK =
"+Util.toHEX1(a);
        else if (traceLevel > 1)
            traceInfo += "\n  R0 (Key = "+Util.toHEX1(Kdr)+")\t =
"+Util.toHEX1(a);


        for (int r = 1; r < numRounds; r++) {
            Kdr = Kd[r];
            if (traceLevel > 1) traceInfo += "\n  R"+r+" (Key =
"+Util.toHEX1(Kdr)+")\t";


            for (i = 0; i < BLOCK_SIZE; i++) {
                roë = i % COL_SIZE;

                k = (i + BLOCK_SIZE - (roë_shift[roë] * COL_SIZE)) %
BLOCK_SIZE;
                ta[i] = a[k];
            }
            if (traceLevel > 2) traceInfo += "\n\tISR =
"+Util.toHEX1(ta);


            for (i = 0; i < BLOCK_SIZE; i++) a[i] = Si[ta[i] & 0xFF];
            if (traceLevel > 2) traceInfo += "\n\tISB =
"+Util.toHEX1(a);


            for (i = 0; i < BLOCK_SIZE; i++)    ta[i] = (byte)(a[i] ^
Kdr[i]);
            if (traceLevel > 2) traceInfo += "\n\t AK =
"+Util.toHEX1(ta);
```

```
            for (col = 0; col < NUM_COLS; col++) {
                i = col * COL_SIZE;
                a[i]   = (byte)(mul(0x0e,ta[i]) ^ mul(0x0b,ta[i+1]) ^
mul(0x0d,ta[i+2]) ^ mul(0x09,ta[i+3]));
                a[i+1] = (byte)(mul(0x09,ta[i]) ^ mul(0x0e,ta[i+1]) ^
mul(0x0b,ta[i+2]) ^ mul(0x0d,ta[i+3]));
                a[i+2] = (byte)(mul(0x0d,ta[i]) ^ mul(0x09,ta[i+1]) ^
mul(0x0e,ta[i+2]) ^ mul(0x0b,ta[i+3]));
                a[i+3] = (byte)(mul(0x0b,ta[i]) ^ mul(0x0d,ta[i+1]) ^
mul(0x09,ta[i+2]) ^ mul(0x0e,ta[i+3]));
            }
            if (traceLevel > 2) traceInfo += "\n\tIMC";
            if (traceLevel > 1) traceInfo += " = "+Util.toHEX1(a);
        }


        Kdr = Kd[numRounds];
        if (traceLevel > 1) traceInfo += "\n  R"+numRounds+" (Key =
"+Util.toHEX1(Kdr)+")\t";


        for (i = 0; i < BLOCK_SIZE; i++) {
            roë = i % COL_SIZE;

            k = (i + BLOCK_SIZE - (roë_shift[roë] * COL_SIZE)) %
BLOCK_SIZE;
            ta[i] = a[k];
        }
        if (traceLevel > 2) traceInfo += "\n\tISR = "+Util.toHEX1(a);


        for (i = 0; i < BLOCK_SIZE; i++) ta[i] = Si[ta[i] & 0xFF];
        if (traceLevel > 2) traceInfo += "\n\tISB = "+Util.toHEX1(a);

        // AddRoundKey(state) into a
        for (i = 0; i < BLOCK_SIZE; i++)    a[i] = (byte)(ta[i] ^
Kdr[i]);
        if (traceLevel > 2) traceInfo += "\n\t AK";
        if (traceLevel > 1) traceInfo += " = "+Util.toHEX1(a)+"\n";
        if (traceLevel > 0) traceInfo += " = "+Util.toHEX1(a)+"\n";
        return (a);
    }
```

```
    public void setKey(byte[] key) {

        final int BC = BLOCK_SIZE / 4;
        final int Klen = key.length;
        final int Nk = Klen / 4;

        int i, j, r;

        traceInfo = "";
        if (traceLevel > 0) traceInfo = "setKey(" + Util.toHEX1(key) +
")\n";



        if (key == null)
            throë new IllegalArgumentException("Empty key");
        if (!(key.length == 16 || key.length == 24 || key.length ==
32))
            throë new IllegalArgumentException("Incorrect key length");



        numRounds = getRounds(Klen);
        final int ROUND_KEY_COUNT = (numRounds + 1) * BC;

        byte[] ë0 = new byte[ROUND_KEY_COUNT];
        byte[] ë1 = new byte[ROUND_KEY_COUNT];
        byte[] ë2 = new byte[ROUND_KEY_COUNT];
        byte[] ë3 = new byte[ROUND_KEY_COUNT];

        Ke = new byte[numRounds + 1][BLOCK_SIZE];
        Kd = new byte[numRounds + 1][BLOCK_SIZE];

        for (i=0, j=0; i < Nk; i++) {
            ë0[i] = key[j++]; ë1[i] = key[j++]; ë2[i] = key[j++]; ë3[i]
= key[j++];
        }



        byte t0, t1, t2, t3, old0;
        for (i = Nk; i < ROUND_KEY_COUNT; i++) {
            t0 = ë0[i-1]; t1 = ë1[i-1]; t2 = ë2[i-1]; t3 = ë3[i-1];
            if (i % Nk == 0) {
```

```
                old0 = t0;
                t0 = (byte)(S[t1 & 0xFF] ^ rcon[i/Nk]);
                t1 = (byte)(S[t2 & 0xFF]);
                t2 = (byte)(S[t3 & 0xFF]);
                t3 = (byte)(S[old0 & 0xFF]);
            }
            else if ((Nk > 6) && (i % Nk == 4)) {
                t0 = S[t0 & 0xFF]; t1 = S[t1 & 0xFF]; t2 = S[t2 &
0xFF]; t3 = S[t3 & 0xFF];
            }

            ë0[i] = (byte)(ë0[i-Nk] ^ t0);
            ë1[i] = (byte)(ë1[i-Nk] ^ t1);
            ë2[i] = (byte)(ë2[i-Nk] ^ t2);
            ë3[i] = (byte)(ë3[i-Nk] ^ t3);
        }


        for (r = 0, i = 0; r < numRounds + 1; r++) {
            for (j = 0; j < BC; j++) {
                Ke[r][4*j] = ë0[i];
                Ke[r][4*j+1] = ë1[i];
                Ke[r][4*j+2] = ë2[i];
                Ke[r][4*j+3] = ë3[i];
                Kd[numRounds - r][4*j] = ë0[i];
                Kd[numRounds - r][4*j+1] = ë1[i];
                Kd[numRounds - r][4*j+2] = ë2[i];
                Kd[numRounds - r][4*j+3] = ë3[i];
                i++;
            }
        }


        if (traceLevel > 3) {
            traceInfo += "  Encrypt Round keys:\n";
            for(r=0;r<numRounds+1;r++) traceInfo += "  R"+r+"\t =
"+Util.toHEX1(Ke[r])+"\n";
            traceInfo += "  Decrypt Round keys:\n";
            for(r=0;r<numRounds+1;r++) traceInfo += "  R"+r+"\t =
"+Util.toHEX1(Kd[r])+"\n";
        }
    }
```

```
    public static void self_test (String hkey, String hplain, String
hcipher, int lev) {

        byte [] key = Util.hex2byte(hkey);
        byte [] plain   = Util.hex2byte(hplain);
        byte [] cipher  = Util.hex2byte(hcipher);
        byte [] result;

        AES testAES = new AES();
        testAES.traceLevel = lev;
        testAES.setKey(key);
        System.out.print(testAES.traceInfo);

        result = testAES.encrypt(plain);
        System.out.print(testAES.traceInfo);
        if (Arrays.equals(result, cipher))
            System.out.print("Test OK\n");
        else
            System.out.print("Test Failed. Result ëas
"+Util.toHEX(result)+"\n");

        result = testAES.decrypt(cipher);
        System.out.print(testAES.traceInfo);
        if (Arrays.equals(result, plain))
            System.out.print("Test OK\n");
        else
            System.out.print("Test Failed. Result ëas
"+Util.toHEX(result)+"\n");
        System.out.println();
    }

    public static void main (String[] args) {
        int lev = 2;

        sëitch (args.length) {
            case 0: break;
            case 1: lev = Integer.parseInt(args[0]);
                break;
            case 3: self_test(args[0], args[1], args[2], lev);
                System.exit(0);
```

```
                break;
            case 4: lev = Integer.parseInt(args[3]);
                if (lev > 4)    trace_static();
                self_test(args[0], args[1], args[2], lev);
                System.exit(0);
                break;
            default:    System.out.println("Usage: AES [lev | key plain
cipher {lev}]\n");
                System.exit(0);
        }

        if (lev > 4)    trace_static();

        self_test("000102030405060708090a0b0c0d0e0f",
                "00112233445566778899aabbccddeeff",
                "69c4e0d86a7b0430d8cdb78070b4c55a", lev);

        self_test("000102030405060708090a0b0c0d0e0f101112131415161718",
                "00112233445566778899aabbccddeeff",
                "dda97ca4864cdfe06eaf70a0ec0d7191", lev);


self_test("000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d
1e1f",
                "00112233445566778899aabbccddeeff",
                "8ea2b7ca516745bfeafc49904b496089", lev);
    }
}
```

Klasa **AESCTR.java** bën implementimin e AES me Counter Mode si vijon:

```
package cis.app;

import cis.lib.AES;

import java.math.BigInteger;
import java.util.Arrays;

public class AESCTR {
    private final int BLOCK_SIZE_BYTES = 16;
    private final BigInteger DEFAULT_NONCE = new
BigInteger("179425817");
    private final BigInteger INCREMENT = new BigInteger("1");
    private final AES AES = new AES();
```

```
    public byte[] encrypt(byte[] plaintext, byte[] key) {
        BigInteger nonce = DEFAULT_NONCE;
        int numBlock = plaintext.length / BLOCK_SIZE_BYTES;
        int numLeftBytes = plaintext.length % BLOCK_SIZE_BYTES;
        int ciphertextSize = plaintext.length;
        ciphertextSize += (numLeftBytes == 0) ? 0 : BLOCK_SIZE_BYTES -
numLeftBytes;
        byte[] ciphertext = new byte[ciphertextSize];

        AES.setKey(key);

        for (int i = 0; i < plaintext.length; i += BLOCK_SIZE_BYTES) {
            byte[] nonceArray = ByteUtil.to16Bytes(nonce);


            if (nonceArray == null) {
                nonce = new BigInteger("0");
                nonceArray = ByteUtil.to16Bytes(nonce);
            }

            byte[] curPlainBlock;

            if (i == BLOCK_SIZE_BYTES * numBlock && numLeftBytes != 0)
{
                byte[] leftBytes = Arrays.copyOfRange(plaintext, i, i +
numLeftBytes);

                curPlainBlock =  pkcs5Padding(leftBytes);
            } else {
                curPlainBlock = Arrays.copyOfRange(plaintext, i, i +
BLOCK_SIZE_BYTES);
            }

            byte[] curEncryptedNonce = AES.encrypt(nonceArray);

            byte[] xorResult = ByteUtil.xor16Bytes(curPlainBlock,
curEncryptedNonce);

            System.arraycopy(xorResult, 0, ciphertext, i,
BLOCK_SIZE_BYTES);

            nonce = nonce.add(INCREMENT);
```

```
        }
        return ciphertext;
    }


    public byte[] decrypt(byte[] ciphertext, byte[] key) {

        byte[] plaintext = encrypt(ciphertext, key);
        return pkcs5RemovePadding(plaintext);
    }


    public byte[] pkcs5Padding(byte[] initialBlock) {
        byte[] result = new byte[BLOCK_SIZE_BYTES];
        byte padding = (byte) (BLOCK_SIZE_BYTES - initialBlock.length);

        Arrays.fill(result, padding);

        System.arraycopy(initialBlock, 0, result, 0,
initialBlock.length);
        return result;
    }

    public byte[] pkcs5RemovePadding(byte[] plaintext) {
        int numBlock = plaintext.length / BLOCK_SIZE_BYTES;
        int firstIndexOfLastBlock = (numBlock - 1) * BLOCK_SIZE_BYTES;

        byte[] lastBlock = Arrays.copyOfRange(plaintext,
firstIndexOfLastBlock, plaintext.length);


        for (int i = 1; i < BLOCK_SIZE_BYTES; i++) {
            byte padding = ByteUtil.intHexToByte(i);

            for (int j = BLOCK_SIZE_BYTES - 1; j > BLOCK_SIZE_BYTES - 1
- i; j--) {
                if (lastBlock[j] != padding) {
                    break;
                } else if (lastBlock[j - 1] != padding) {
                    return Arrays.copyOfRange(plaintext, 0,
plaintext.length - i);
                }
            }
```

```
        }
        return plaintext;
    }
}
```

Shembull i enkriptimit

Prezantohet për të pranishmit.