UNIVERSITETI I PRISHTINES FAKULTETI I SHKENCAVE MATEMATIKO-NATYRORE

Tema: Hash Message Authentication Code (HMAC)

Lënda: Siguri e të dhënave

Punuar nga:

Ardit Maliqi
Atlant Klaiqi
Andi Alidema

Ligjëruesit e lëndës:

Prof. Artan Berisha
Asst.Besnik Duriqi

Siguria e Mesazheve me Kode Verifikimi të Hashit (HMAC) është një mekanizëm i përdorur për verifikimin e autentifikimit dhe integritetit të mesazheve. Përfshin përdorimin e funksioneve kriptografike të hashit për të prodhuar një vlerë të caktuar të hashit ose rezumet e një mesazhi, të kombinuar me një çelës sekret. HMAC-u u përshkrua për herë të parë në vitin 1996 nga Mihir Bellare, Ran Canetti dhe Hugo Kraëczyk.

Për të kuptuar kontekstin e HMAC-ut, është e rëndësishme të njihni funksionet e hashit. Një funksion hash merr një hyrje (mesazh) dhe prodhon një dalje të caktuar (vlerë hash), që zakonisht është një rezumë e hashit. Tiparet kryesore të një funksioni hash janë se është deterministik (një hyrje e njëjtë do të prodhojë gjithmonë një dalje të njëjtë) dhe se është e pamundur kompjuterikisht të marrësh hyrjen origjinale nga vlera e saj hash.

HMAC-u u zhvillua për të siguruar një metodë të sigurt për të verifikuar integritetin dhe autenticitetin e mesazheve të shkëmbyera mes dy palëve, edhe nëse mesazhet transmetohen nëpërmjet një kanali të padësigurt. Ai adreson disa vulnerabilitete në metodën e mëparshme të njohur si Keyed-Hash Message Authentication Code (KHMAC).

HMAC arrin këtë duke kombinuar mesazhin me një çelës sekret dhe duke zbatuar një funksion hash në rezultatin. Vlera e hashit rezultuese përdoret si etiketë autentifikimi për mesazhin. Marrësi pastaj mund të llogarisë HMAC-në pavarësisht duke përdorur çelësin sekret të ndarë dhe ta krahasojë atë me etiketën e pranuar. Nëse HMAC-u i llogaritur përputhet me etiketën e pranuar, tregon se mesazhi nuk është ndryshuar dhe është autentik.

HMAC-u ofron disa avantazhe të sigurisë. Parandalon ndryshimet e paautorizuara në mesazh, pasi çdo ndryshim në mesazh do të rezultojë në një HMAC tjetër. Ai gjithashtu siguron integritetin e mesazhit duke verifikuar se dërguesi posedon çelësin sekret të ndarë të përdorur për të llogaritur HMAC-un.

HMAC-u mund të zbatohet duke përdorur funksione të hashit të ndryshme, siç janë MD5, SHA-1, SHA-256 dhe të tjera. Megjithatë, rekomandohet përdorimi i funksioneve të hashit më të fortë si SHA-256 ose SHA-3 në aplikacionet moderne për shkak të vulnerabiliteteve të zbuluara në funksionet e hashit më të vjetra.

Në përgjithësi, HMAC-u është bërë një metodë e gjerësisht e përdorur për të siguruar integritetin dhe autenticitetin e mesazheve në aplikacione të ndryshme, duke përfshirë protokollet e rrjetit, nënshkrimet dixhitale dhe kanale të sigurta të komunikimit. Dizajni i tij ofron një zgjidhje të fortë dhe efikase për mbrojtjen e integritetit të të dhënave dhe sigurimin e autenticitetit të mesazheve.

Avantazhet e Hash Message Authentication Code (HMAC):

- ➢ Integriteti i Mesazhit: HMAC siguron një siguri të fortë për integritetin e mesazhit. Duke kombinuar mesazhin me një çelës sekret dhe duke aplikuar një funksion hash, çdo modifikim ose ndryshim në mesazh do të rezultojë në një vlerë HMAC të ndryshme. Kjo lejon pranuesin të verifikojë nëse mesazhi është ndryshuar.
- ➢ Autentifikimi: HMAC mundëson autentifikimin e mesazhit duke verifikuar se dërguesi posedon çelësin sekret të ndarë. Vetëm dërguesi me çelësin korrekt mund të prodhojë vlerën HMAC korrekte. Kjo siguron që mesazhi ka origjinë nga një burim autentik.
- ➢ Siguria e bazuar në çelësat: HMAC përdor një çelës sekret që dihet vetëm nga dërguesi dhe pranuesi. Kjo përmirëson sigurinë e procesit të autentifikimit, pasi një sulmues pa njohuri të çelësit sekret nuk mund të gjenerojë një HMAC të vlefshme.
- ➢ Efikasiteti: HMAC është efikas në kalkulimin kompjuterik. Përdor funksione hash, të cilat janë të projektuara për të kalkuluar shpejtë vlerat hash. Kjo bën HMAC-un të përshtatshëm për aplikacione në kohë reale ku efikasiteti është i rëndësishëm.
- ➢ Aplikim gjerë: HMAC është përdorur gjerësisht dhe është integruar në protokolle dhe sisteme kriptografike të ndryshme. Është i përkrahur nga shumë gjuhë programimi dhe libra kriptografikë, duke e bërë të gatshëm për zbatim në aplikacione të ndryshme.

Disavantazhet e HMAC:

- ➢ Menaxhimi i çelësave: Siguria e HMAC varet fort nga menaxhimi i çelësit sekret. Shpërndarja e duhur e çelësit, ruajtja dhe mbrojtja janë të domosdoshme për të parandaluar qasjen ose zbulimin e padëshiruar. Menaxhimi i çelësave mund të jetë kompleks dhe sfidues, veçanërisht në sisteme me shkallë të madhe.
- ➢ Pikë e vetme e dështimit: Nëse çelësi sekret i përdorur në HMAC kompromentohet, ky fakt mund të kompromentojë sigurinë e të gjitha mesazheve që janë autentifikuar

me atë çelës. Prandaj, është e rëndësishme të mbrohet sekretësia e çelësit për të ruajtur integritetin dhe autenticitetin e mesazheve.

- ➢ Autentifikim i kufizuar: HMAC ofron vetëm autentifikim dhe verifikim të integritetit. Nuk ofron konfidencialitet, dmth përmbajtja e mesazhit ende mund të lexohet nga një sulmues nëse kapet. Masat shtesë si shifruesi duhet të përdoren së bashku me HMAC për të siguruar konfidencialitetin.
- ➢ Vulnerabilitet ndaj algoritmeve: Siguria e HMAC varet nga funksioni hash nënligjor i përdorur. Është e rëndësishme të zgjidhni një funksion hash të sigurt dhe të qëndrueshëm. Funksionet hash më të dobëta, si MD5 ose SHA-1, kanë vulnerabilitete të njohura dhe duhet të shmangen.
- ➢ Sulmet e ringjalljes: HMAC vetë nuk mund të mbrojë kundër sulmeve të ringjalljes, ku një sulmues kap dhe ridërgon mesazhe të vlefshme të mbrojtura me HMAC. Masat shtesë si markat e kohës ose numrat e sekuenës duhet të përdoren për të parandaluar sulmet e ringjalljes.
- ➢ Është e rëndësishme të theksohet se, pavarësisht disavantazheve të përmendura, HMAC mbetet një mekanizëm i gjerësisht përdorur dhe efektiv për autentifikimin e mesazheve dhe verifikimin e integritetit kur përdoret në mënyrë të përshtatshme dhe në kombinim me masa të tjera të sigurisë.

Algoritmi për Hash Message Authentication Code (HMAC) përfshin disa hapa:

1) Zgjidh një funksion hash kriptografik: Zgjidh një funksion hash të sigurt, siç janë SHA-256 ose SHA-3. Ky funksion hash do të përdoret për të llogaritur HMAC-in.
2) Përgatitja e çelësit: Nëse çelësi sekret i dhënë është më i gjatë se madhësia e bllokut të funksionit hash të zgjedhur, aplikoni funksionin hash në çelës. Nëse është më i shkurtër, shtoni zero tek çelësi derisa të arrijë madhësinë e bllokut. Në vend të kësaj, mund të përdorni një funksion për nxjerrjen e çelësit (KDF) për të prodhuar një çelës me gjatësinë e dëshiruar.
3) XOR padding i çelësit: XOR secilin bajt të çelësit me vlerën 0x36. Kjo quhet paddingu i brendshëm.
4) Shtoni paddingun e brendshëm: Shtoni paddingun e brendshëm në fillim të mesazhit.

5) Llogaritni hashin e brendshëm: Aplikoni funksionin hash të zgjedhur në rezultatin e fituar në hapin 4.
6) XOR paddingu i çelësit (përsëri): XOR secilin bajt të çelësit me vlerën 0x5C. Kjo quhet paddingu i jashtëm.
7) Shtoni paddingun e jashtëm: Shtoni paddingun e jashtëm në rezultatin e fituar në hapin 5.
8) Llogaritni hashin e jashtëm: Aplikoni funksionin hash të zgjedhur në rezultatin e fituar në hapin 7.
9) Rezultati përfundimtar: HMAC-i përfundimtar është output-i i fituar në hapin 8. Zakonisht është paraqitur si një vlerë hash me madhësi të fiksuar.

Algoritmi HMAC kombinon çelësin dhe mesazhin në një mënyrë të veçantë, duke përfshirë paddingun e brendshëm dhe të jashtëm për të përmirësuar sigurinë. Duke aplikuar funksionin hash dy herë duke përdorur vlera të ndryshme paddingu, HMAC siguron një metodë të sigurtë dhe të qëndrueshme për autentifikimin e mesazheve dhe verifikimin e integritetit.

Kodi ne java

Klasa kryesore:

```
public class HMAC_SSHA_1 {



    public String hmac_ssha_1(boolean [] k, String m )

    {




        Shendrrimet s=new Shendrrimet();

        Simplified_SHA_1 sa=new Simplified_SHA_1();

        m=s.strToBinary(m);



        // llogaritja e k'

        boolean [] k1=new boolean[32];

        if(k.length==k1.length)

        {
```

```
            for(int i=0; i<k1.length; i++)

            {k1[i]=k[i];}

        }

        else{

            String b=s.boolean_string(k);

            String hash_b=sa.hash(b);

            k1=s.hex_boolean(hash_b);

        }

            // ipad 0x36   00110110 perseritet 4 here per te arritur
gjatesine e bllokut 32.

                                                        boolean       []
ipad={false,false,true,true,false,true,true,false,false,false,true,true
,false,true,true,false,false,false,true,true,false,true,true,false,fals
e,false,true,true,false,true,true,false};



            // opad 0x5C   01011100 perseritet 4 here per te arritur
gjatesine e bllokut 32.

                                                        boolean       []
opad={false,true,false,true,true,true,false,false,false,true,false,true
,true,true,false,false,false,true,false,true,true,true,false,false,fals
e,true,false,true,true,true,false,false};



        // rez pas K' xor opad

        String k_opad=xor_to_String(k1, opad);



        //String per (K'xor opad)|| m

        String k_opad_m=k_opad+m;
```

```
        // H((K'xor opad)|| m)
        String h_k_opad_m=s.hex_binary(sa.hash(k_opad_m));

        // K' xor ipad
        String k_ipad=xor_to_String(k1, ipad);

        //  (K' xor ipad) || H((K'xor opad)|| m)
        String to_hmac=k_ipad+h_k_opad_m;

        //  H((K' xor ipad) || H((K'xor opad)|| m))
        String hmac_ssha_1=sa.hash(to_hmac);

        return hmac_ssha_1;
    }

    public static String xor_to_String(boolean[] a, boolean[] b)
    {
        boolean[] c=new boolean[a.length];
        for(int i=0; i<c.length; i++)
        {
            c[i]=a[i]^c[i];
        }
        String cc="";
        for(int i=0; i<c.length; i++)
        {
```

```
            if(c[i])

            {cc=cc+"1";}

            else{cc=cc+"0";}

        }



        return cc;

    }



}
```

Klasa Simplified_SHA1

```
public class Simplified_SHA_1{
    static boolean[]
W0,W1,W2,W3,W4,W5,W6,W7,W8,W9,W10,W11,W12,W13,W14,W15;

    public String hash(String h)
    {   String vektori_inicializues="45AFACFE"; // A=45; B=AF; C=CF;
D=FE;
        String hash_vlera="";
        String hash1=paddedString(h);
        int r=hash1.length()/8;
        for(int i=0; i<r; i++)
        {
            hash_vlera=H(hash1.substring(i*8,
i*8+8),vektori_inicializues);
            vektori_inicializues=hash_vlera;
        }
        return hash_vlera;
    }

    public  String paddedString(String mesazhi)
    {   String padded_string="";
        int l=mesazhi.length();
        int k=32-16-(l+1);
        while(k<0)
```

```
        {
            k=32+k;
        }
        k=k%32;
        padded_string=mesazhi+"1";
        for(int i=0; i<k; i++)
        {padded_string=padded_string+"0";}
        Shendrrimet s=new Shendrrimet();
        padded_string=padded_string+s.int_to_bin(l);

        // shendrrimi i mesazhit ne hex
         padded_string=s.bin_to_hex(padded_string);
        return padded_string;

    }

    public static String H(String a, String iv)
    {   boolean[] x_i=String_bin(a);
        kalkulimi_Ws(x_i);
        char[] ABCD=new char[8];
        for(int i=0; i<ABCD.length; i++)
        {
            ABCD[i]=iv.charAt(i);
        }
        Shendrrimet s=new Shendrrimet();
        boolean[] K1=s.hex_bin('5','A');// K1=5A
        boolean[] K2=s.hex_bin('E','7');// K2=E7
        boolean[] K3=s.hex_bin('8','C');// K3=8C
        boolean[] K4=s.hex_bin('B','D');// K4=BD
        char[] etapa1=etapa(1,W0,W1,W2,W3,ABCD,K1); // parametri i pare
tregon funksionin f1
        char[] etapa2=etapa(2,W4,W5,W6,W7,etapa1,K2);
        char[] etapa3=etapa(3,W8,W9,W10,W11,etapa2,K3);
        char[] etapa4=etapa(4,W12,W13,W14,W15,etapa3,K4);

        boolean[] A_fillim=s.hex_bin(ABCD[0],ABCD[1]);
        boolean[] B_fillim=s.hex_bin(ABCD[2],ABCD[3]);
        boolean[] C_fillim=s.hex_bin(ABCD[4],ABCD[5]);
        boolean[] D_fillim=s.hex_bin(ABCD[6],ABCD[7]);

        boolean[] A_fund=s.hex_bin(etapa4[0],etapa4[1]);
        boolean[] B_fund=s.hex_bin(etapa4[2],etapa4[3]);
        boolean[] C_fund=s.hex_bin(etapa4[4],etapa4[5]);
```

```
        boolean[] D_fund=s.hex_bin(etapa4[6],etapa4[7]);

        char[] Am=s.bin_hex(mbledhja_mod(A_fillim,A_fund));
        char[] Bm=s.bin_hex(mbledhja_mod(B_fillim,B_fund));
        char[] Cm=s.bin_hex(mbledhja_mod(C_fillim,C_fund));
        char[] Dm=s.bin_hex(mbledhja_mod(D_fillim,D_fund));

        char[] hash=new char[8];
        hash[0]=Am[0];
        hash[1]=Am[1];
        hash[2]=Bm[0];
        hash[3]=Bm[1];
        hash[4]=Cm[0];
        hash[5]=Cm[1];
        hash[6]=Dm[0];
        hash[7]=Dm[1];

        String enkriptimi="";
        for(int i=0; i<8; i++)
           {enkriptimi=enkriptimi+hash[i];}

        return enkriptimi;
    }

    public static char[] etapa(int funksioni_i_n,boolean[] w_1,
boolean[] w_2, boolean[] w_3, boolean[] w_4, char[] vlerat_fillestare,
boolean[] K )
    {
        char[] etapa_n=new char[8];
        etapa_n=roundi(vlerat_fillestare,w_1,K,funksioni_i_n);
        etapa_n=roundi(etapa_n,w_2,K,funksioni_i_n);
        etapa_n=roundi(etapa_n,w_3,K,funksioni_i_n);
        etapa_n=roundi(etapa_n,w_4,K,funksioni_i_n);
        return etapa_n;
    }

    public static char [] roundi (char[] a,boolean[] w, boolean[] K,
int funksioni_i_n)
    {
        /* Sqarim: per shkak te implementimit te klases Shendrrimet
input merret varg i karaktereve ashtu qe
        A=a[0]a[1], pra secili nr heksadecimal(paraqitet me a[i]). Meqe
nga skema A eshte 8 bit atehere merr dy karaktere
```

```
        nga vargu. Kjo metode kthen varg te karaktereve me gjatesi 8
(nga dy per secilen:A,B,C,D). Punohet me vargje
        booleane */
        Shendrrimet s=new Shendrrimet();
        boolean [] A=s.hex_bin(a[0],a[1]);
        boolean [] B=s.hex_bin(a[2],a[3]);
        boolean [] C=s.hex_bin(a[4],a[5]);
        boolean [] D=s.hex_bin(a[6],a[7]);

        // Vlerat direkte A->B, C->D
        char[] B_1=s.bin_hex(A);
        char[] D_1=s.bin_hex(C);

        // Vlera B->C por pas shift B<<<7(mund te merret edhe si >>>1)
        boolean[] B_shift=new boolean[8];
        for(int i=1; i<8; i++)
        {  B_shift[i]=B[i-1];}
           B_shift[0]=B[7];
        char[] C_1=s.bin_hex(B_shift);

        // Llogaritja e A
        boolean [] A_shift=new boolean[8];  // shift i A <<<3
        for(int i=0; i<5; i++)
        {  A_shift[i]=A[i+3];}
           A_shift[5]=A[0];
           A_shift[6]=A[1];
           A_shift[7]=A[2];
        boolean[] llogaritja_A=new boolean[8];
        llogaritja_A=mbledhja_mod(D,f(B,C,funksioni_i_n));
        llogaritja_A=mbledhja_mod(llogaritja_A,A_shift);
        llogaritja_A=mbledhja_mod(llogaritja_A,w);
        llogaritja_A=mbledhja_mod(llogaritja_A,K);

        char[] A_1=s.bin_hex(llogaritja_A);


        // nga dy karaktere paraqesin dy vlerat heksadecimale per
A,B,C,D
        char[] A_B_C_D=new char[8];
        A_B_C_D[0]=A_1[0];
        A_B_C_D[1]=A_1[1];
        A_B_C_D[2]=B_1[0];
        A_B_C_D[3]=B_1[1];
```

```
        A_B_C_D[4]=C_1[0];
        A_B_C_D[5]=C_1[1];
        A_B_C_D[6]=D_1[0];
        A_B_C_D[7]=D_1[1];


       return A_B_C_D;
    }

    private static void kalkulimi_Ws(boolean[] x_i)
    {
        // ndarja e mesazhit x_i ne kater pjese x_i_j
        boolean[] x_i_0=new boolean[8];
        boolean[] x_i_1=new boolean[8];
        boolean[] x_i_2=new boolean[8];
        boolean[] x_i_3=new boolean[8];
        // mbushja e x_i_j
        for(int k=0; k<8; k++)
        {     x_i_0[k]=x_i[k];
              x_i_1[k]=x_i[k+8];
              x_i_2[k]=x_i[k+16];
              x_i_3[k]=x_i[k+24];    }
        // mbushja e W0-W3 direkte nga x_i_j
        W0=x_i_0;
        W1=x_i_1;
        W2=x_i_2;
        W3=x_i_3;

        // mbushja e W4-16 ashtu qe (Wj-4 XOR Wj-2)<<<2
        W4=w_xor_shift(W0, W2);
        W5=w_xor_shift(W1, W3);
        W6=w_xor_shift(W2, W4);
        W7=w_xor_shift(W3, W5);
        W8=w_xor_shift(W4, W6);
        W9=w_xor_shift(W5, W7);
        W10=w_xor_shift(W6, W8);
        W11=w_xor_shift(W7, W9);
        W12=w_xor_shift(W8, W10);
        W13=w_xor_shift(W9, W11);
        W14=w_xor_shift(W10, W12);
        W15=w_xor_shift(W11, W13);

    }
```

```
    public static boolean [] w_xor_shift(boolean[] w1, boolean[] w2)
    {       boolean[] w1_xor_shift_w2=new boolean[8];
            boolean[] w1_XOR_w2=new boolean[8];
            for(int k=0; k<8; k++)
            {  w1_XOR_w2[k]=w1[k]^w2[k];}
            w1_xor_shift_w2[6]=w1_XOR_w2[0];
            w1_xor_shift_w2[7]=w1_XOR_w2[1];
            for(int k=0; k<6; k++)
            { w1_xor_shift_w2[k]=w1_XOR_w2[k+2]; }

            return w1_xor_shift_w2;
    }

    public static boolean[] f(boolean[] B,boolean[] C,int
funksioni_i_n)
    {   boolean[] funksioni=new boolean[8];
        if(funksioni_i_n==1)
        {    for(int i=0; i<8; i++)
                { funksioni[i]=B[i]&&C[i]; }}

                if(funksioni_i_n==2|| funksioni_i_n==4)
                {    for(int i=0; i<8; i++)
                        { funksioni[i]=B[i]^C[i]; }}

                        if(funksioni_i_n==3)
                           {    for(int i=0; i<8; i++)
                                   { funksioni[i]=B[i]^(!C[i]); }}

        return funksioni;
    }

    public static boolean[] mbledhja_mod(boolean[] a, boolean[] b)
    {
        int aa=0;
        int bb=0;
        for(int k=0; k<8; k++)
        {  if(a[k])
            {aa=aa+(int)(Math.pow(2,7-k));}
           if(b[k])
            {bb=bb+(int)(Math.pow(2,7-k));}}
        int c=aa+bb;
        c=c%(int)(Math.pow(2,8));
```

```
        boolean[] mbetja=new boolean[8];
        for(int i=0; i<8; i++)
        {   if(c%2==1)
               {mbetja[7-i]=true;}
            c=c/2;}

        return mbetja;
    }

    public static boolean[] String_bin(String a)
    {   boolean[] xi=new boolean[32];
        Shendrrimet s=new Shendrrimet();
        boolean[] x0=s.hex_bin(a.charAt(0),a.charAt(1));
        boolean[] x1=s.hex_bin(a.charAt(2),a.charAt(3));
        boolean[] x2=s.hex_bin(a.charAt(4),a.charAt(5));
        boolean[] x3=s.hex_bin(a.charAt(6),a.charAt(7));

        for(int i=0; i<8; i++)
        {   xi[i]=x0[i];
            xi[i+8]=x1[i];
            xi[i+16]=x2[i];
            xi[i+24]=x3[i]; }

        return xi;
    }

}
```