Experiment 13

AIM: Write a Program to implement Cryptographic Hash Functions and Applications (HMAC): to understand the need, design, and applications of collision-resistant hash functions.

Roll No.	37
Name	Mikil Lalwani
Class	D15-B
Subject	Internet Security Lab
LO Mapped	LO6: Demonstrate the network security system using open source tools.

AIM:

Write a Program to implement Cryptographic Hash Functions and Applications (HMAC): to understand the need, design, and applications of collision-resistant hash functions.

THEORY:

HMAC algorithm stands for Hashed or Hash-based Message Authentication Code. It is a result of work on developing a MAC derived from cryptographic hash functions. HMAC is a great resistant to cryptanalysis attacks as it uses the Hashing concept twice. HMAC consists of twin benefits of Hashing and MAC and thus is more secure than any other authentication code.

RFC 2104 has issued HMAC, and HMAC has been made compulsory to implement in IP security. The FIPS 198 NIST standard has also been issued by HMAC.

Objectives –

As the Hash Function, HMAC is also aimed to be one way, i.e, easy to generate output from input but complex the other way around.

It aims at being less affected by collisions than the hash functions.

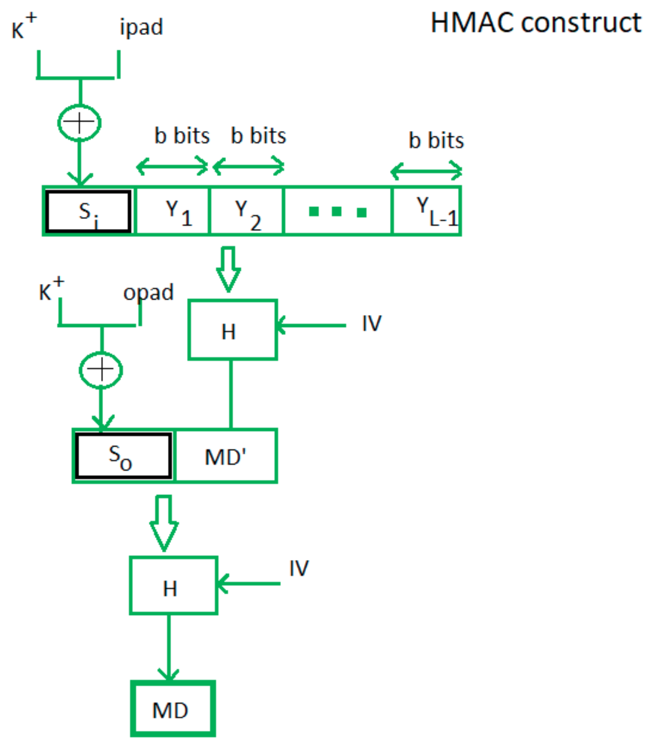
HMAC reuses the algorithms like MD5 and SHA-1 and checks to replace the embedded hash functions with more secure hash functions, in case found.

HMAC tries to handle the Keys in a more simple manner.

HMAC algorithm –

The working of HMAC starts with taking a message M containing blocks of length b bits. An input signature is padded to the left of the message and the whole is given as input to a hash function which gives us a temporary message-digest MD'. MD' again is appended to an output signature and the whole is applied to a hash function again, the result is our final message digest MD.

Here is a simple structure of HMAC:



Here, H stands for Hashing function,

M is the original message

Si and So are input and output signatures respectively,

Yi is the ith block in original message M, where I ranges from [1, L)

L = the count of blocks in M

K is the secret key used for hashing

IV is an initial vector (some constant)

The generation of input signature and output signature Si and So respectively.

$$S_i = K^+ \oplus ipad$$

where K^+ is nothing but K padded with zeros on the left so that the result is b bits in length

$$S_o = K^+ \oplus opad$$

where ipad and opad are 00110110 and 01011100 respectively taken b/8 times repeatedly.

$$MD' = H(S_i \,||\, M)$$

$$MD = H(S_o \,||\, MD') \quad \text{or } MD = H(S_o \,||\, H(S_i \,||\, M))$$

HMAC adds a compression instance to the processing to a normal hash function. This structural implementation holds efficiency for shorter MAC values.

CODE:

```
#include <stdio.h>
#include <windows.h>
#include <wincrypt.h>

int main()
{
   HCRYPTPROV hProv = NULL;
   HCRYPTHASH hHash = NULL;
   HCRYPTKEY hKey = NULL;
   HCRYPTHASH hHmacHash = NULL;
   PBYTE pbHash = NULL;
   DWORD dwDataLen = 0;
   BYTE Data1[] = {0x70, 0x61, 0x73, 0x73, 0x77, 0x6F, 0x72, 0x64};
   BYTE Data2[] = {0x6D, 0x65, 0x73, 0x73, 0x61, 0x67, 0x65};
   HMAC_INFO HmacInfo;

   ZeroMemory(&HmacInfo, sizeof(HmacInfo));
   HmacInfo.HashAlgid = CALG_SHA1;

   if (!CryptAcquireContext(&hProv,NULL,NULL,PROV_RSA_FULL,CRYPT_VERIFYCONTEXT))
   {
      printf(" Error in AcquireContext 0x%08x \n",
           GetLastError());
      goto ErrorExit;
   }

   if (!CryptCreateHash(hProv,CALG_SHA1,0,0,&hHash))
   {
      printf("Error in CryptCreateHash 0x%08x \n",
           GetLastError());
      goto ErrorExit;
   }

   if (!CryptHashData(hHash,Data1,sizeof(Data1),0))
   {
      printf("Error in CryptHashData 0x%08x \n",
           GetLastError());
      goto ErrorExit;
   }

   if (!CryptDeriveKey(hProv,CALG_RC4,hHash,0,&hKey))
   {
      printf("Error in CryptDeriveKey 0x%08x \n",
           GetLastError());
      goto ErrorExit;
   }

   if (!CryptCreateHash(hProv,CALG_HMAC,hKey,0,&hHmacHash))
   {
      printf("Error in CryptCreateHash 0x%08x \n",
           GetLastError());
      goto ErrorExit;
   }

   if (!CryptSetHashParam(hHmacHash,HP_HMAC_INFO,(BYTE *)&HmacInfo,0))
   {
      printf("Error in CryptSetHashParam 0x%08x \n",
           GetLastError());
      goto ErrorExit;
   }

   if (!CryptHashData(hHmacHash,Data2,sizeof(Data2),0))
   {
      printf("Error in CryptHashData 0x%08x \n",
           GetLastError());
      goto ErrorExit;
   }

   if (!CryptGetHashParam(hHmacHash,HP_HASHVAL,NULL,&dwDataLen,
          0))
   {
      printf("Error in CryptGetHashParam 0x%08x \n",
           GetLastError());
      goto ErrorExit;
   }

   pbHash = (BYTE *)malloc(dwDataLen);
   if (NULL == pbHash)
```

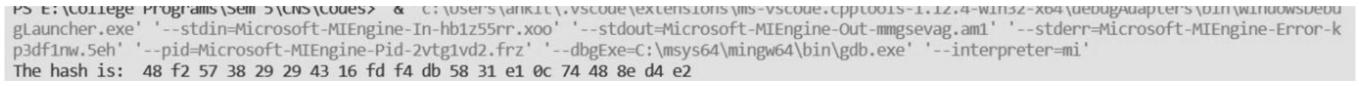
```
    {
        printf("unable to allocate memory\n");
        goto ErrorExit;
    }

    if (!CryptGetHashParam(hHmacHash,HP_HASHVAL,pbHash,&dwDataLen,0))
    {
        printf("Error in CryptGetHashParam 0x%08x \n", GetLastError());
        goto ErrorExit;
    }

    printf("The hash is:  ");
    for (DWORD i = 0; i < dwDataLen; i++)
    {
        printf("%2.2x ", pbHash[i]);
    }
    printf("\n");

ErrorExit:
    if (hHmacHash)
        CryptDestroyHash(hHmacHash);
    if (hKey)
        CryptDestroyKey(hKey);
    if (hHash)
        CryptDestroyHash(hHash);
    if (hProv)
        CryptReleaseContext(hProv, 0);
    if (pbHash)
        free(pbHash);
    return 0;
}
```

OUTPUT:



```
PS E:\College Programs\Sem 5\CNS\Codes>  & 'c:\Users\ankit\.vscode\extensions\ms-vscode.cpptools-1.12.4-win32-x64\debugAdapters\bin\WindowsDebu
gLauncher.exe' '--stdin=Microsoft-MIEngine-In-hb1z55rr.xoo' '--stdout=Microsoft-MIEngine-Out-mmgsevag.am1' '--stderr=Microsoft-MIEngine-Error-k
p3df1nw.5eh' '--pid=Microsoft-MIEngine-Pid-2vtg1vd2.frz' '--dbgExe=C:\msys64\mingw64\bin\gdb.exe' '--interpreter=mi'
The hash is:  48 f2 57 38 29 29 43 16 fd f4 db 58 31 e1 0c 74 48 8e d4 e2
```

CONCLUSION:

We have successfully implemented a program on Cryptographic Hash Functions and Applications (HMAC): and understood the need, design, and applications of collision-resistant hash functions.