Use collision resistant hash function to build H-MACs

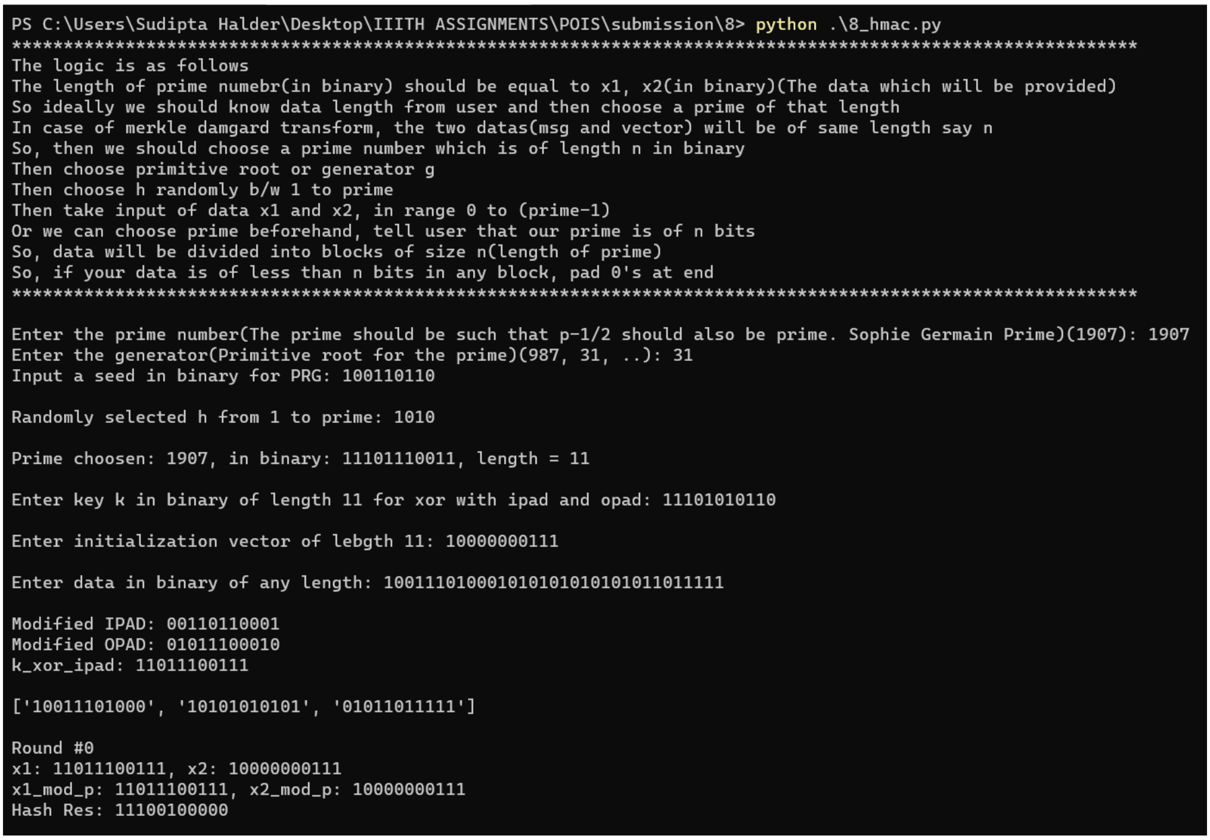
Assumptions: Used previously built PRG to generate h, DLP based hash function as h_s and merkle damgard construction.

Input format:

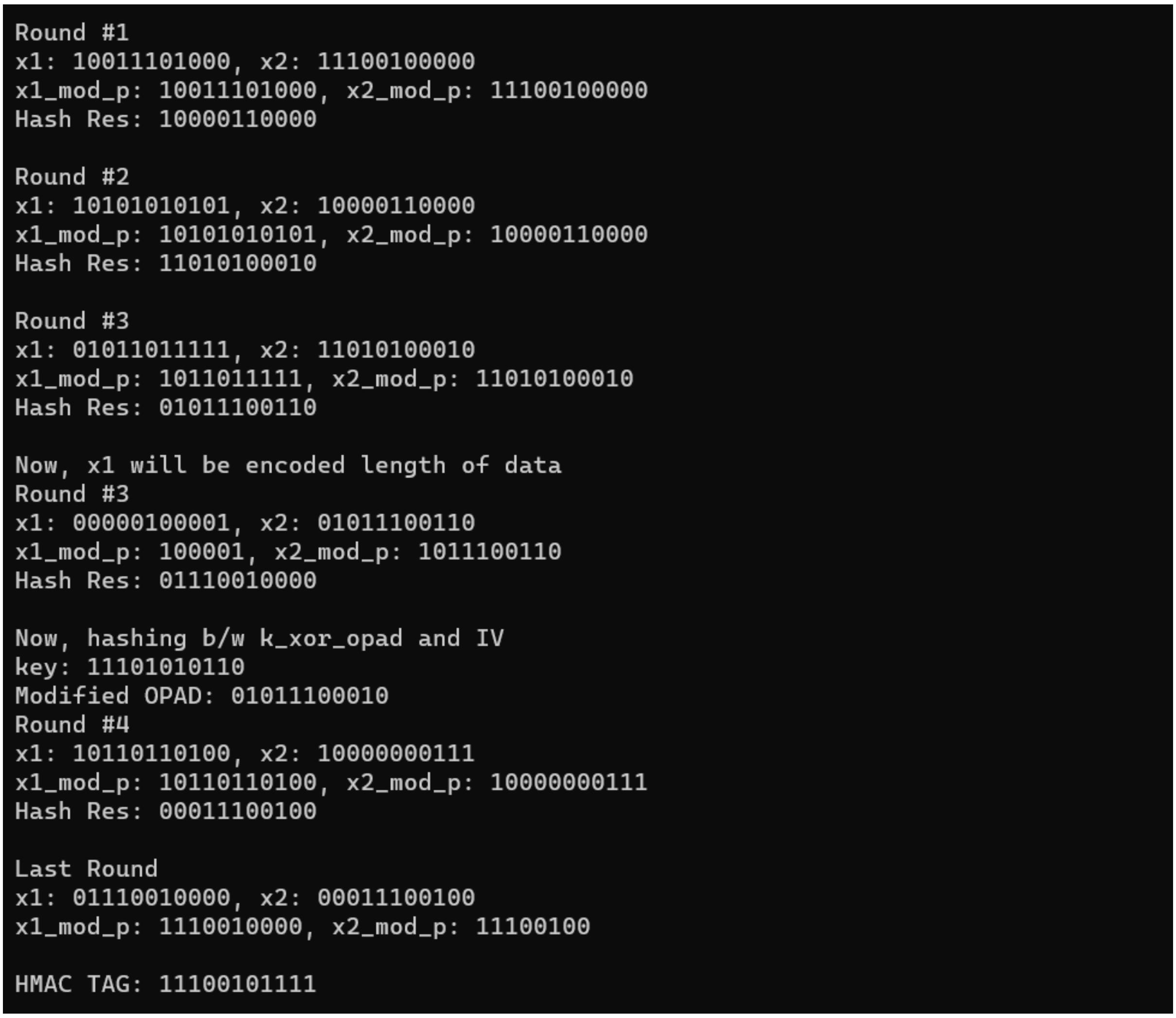
1. It will ask for a safe prime number p in integer format.
2. It will ask for a generator (primitive root) of that prime.
3. It will then ask for a seed to generate h (b/w 1 to prime) via PRG.
4. Then, enter k of length = no of bits in prime.
5. Then, it'll ask to input a initialization vector of length l, where l = no of bits in prime.
6. Then data input of any length.

Output Format:

1. It will output the HMAC TAG.



```
PS C:\Users\Sudipta Halder\Desktop\IIITH ASSIGNMENTS\POIS\submission\8> python .\8_hmac.py
**************************************************************************************************************
The logic is as follows
The length of prime numebr(in binary) should be equal to x1, x2(in binary)(The data which will be provided)
So ideally we should know data length from user and then choose a prime of that length
In case of merkle damgard transform, the two datas(msg and vector) will be of same length say n
So, then we should choose a prime number which is of length n in binary
Then choose primitive root or generator g
Then choose h randomly b/w 1 to prime
Then take input of data x1 and x2, in range 0 to (prime-1)
Or we can choose prime beforehand, tell user that our prime is of n bits
So, data will be divided into blocks of size n(length of prime)
So, if your data is of less than n bits in any block, pad 0's at end
**************************************************************************************************************

Enter the prime number(The prime should be such that p-1/2 should also be prime. Sophie Germain Prime)(1907): 1907
Enter the generator(Primitive root for the prime)(987, 31, ..): 31
Input a seed in binary for PRG: 100110110

Randomly selected h from 1 to prime: 1010

Prime choosen: 1907, in binary: 11101110011, length = 11

Enter key k in binary of length 11 for xor with ipad and opad: 11101010110

Enter initialization vector of lebgth 11: 10000000111

Enter data in binary of any length: 100111010001010101010101011011111

Modified IPAD: 00110110001
Modified OPAD: 01011100010
k_xor_ipad: 11011100111

['10011101000', '10101010101', '01011011111']

Round #0
x1: 11011100111, x2: 10000000111
x1_mod_p: 11011100111, x2_mod_p: 10000000111
Hash Res: 11100100000
```



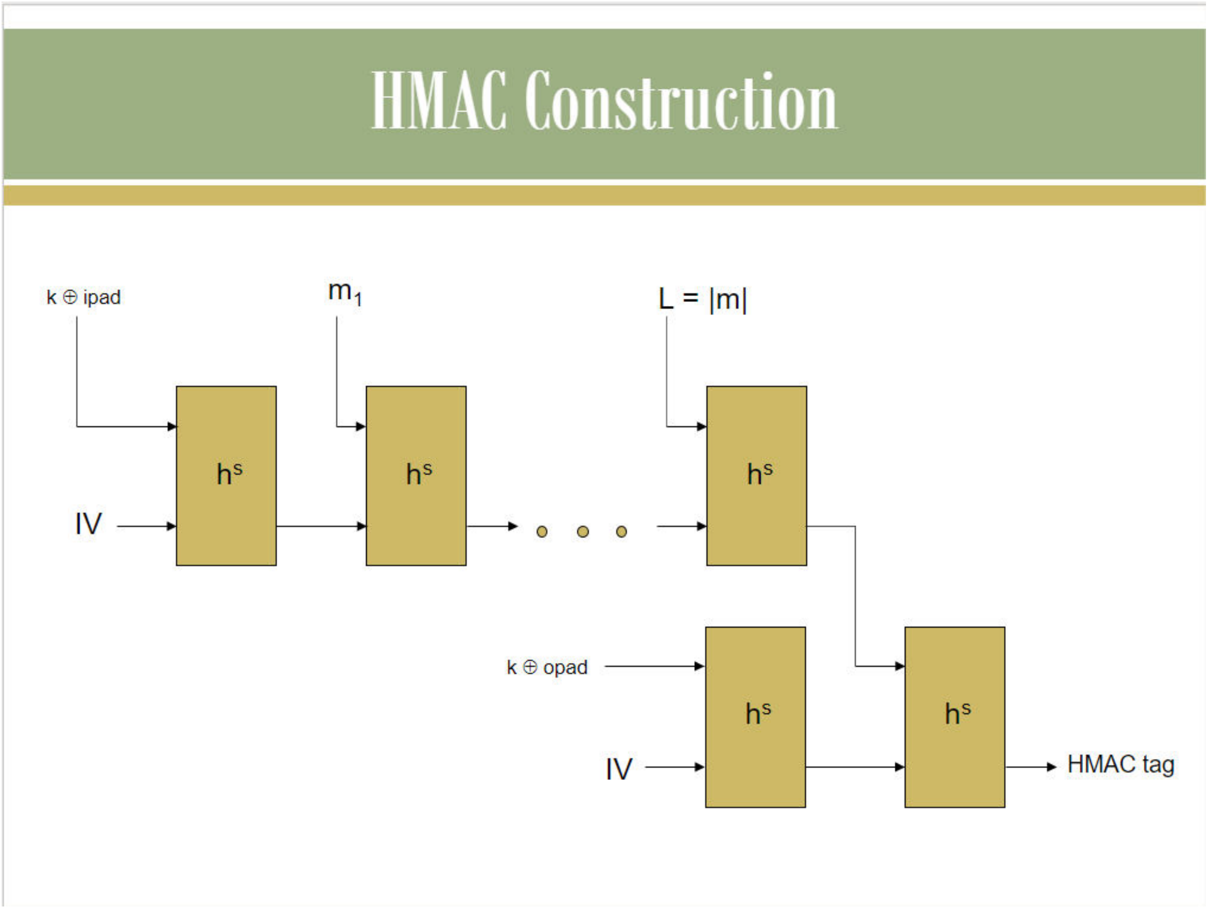
```
Round #1
x1: 10011101000, x2: 11100100000
x1_mod_p: 10011101000, x2_mod_p: 11100100000
Hash Res: 10000110000

Round #2
x1: 10101010101, x2: 10000110000
x1_mod_p: 10101010101, x2_mod_p: 10000110000
Hash Res: 11010100010

Round #3
x1: 01011011111, x2: 11010100010
x1_mod_p: 1011011111, x2_mod_p: 11010100010
Hash Res: 01011100110

Now, x1 will be encoded length of data
Round #3
x1: 00000100001, x2: 01011100110
x1_mod_p: 100001, x2_mod_p: 1011100110
Hash Res: 01110010000

Now, hashing b/w k_xor_opad and IV
key: 11101010110
Modified OPAD: 01011100010
Round #4
x1: 10110110100, x2: 10000000111
x1_mod_p: 10110110100, x2_mod_p: 10000000111
Hash Res: 00011100100

Last Round
x1: 01110010000, x2: 00011100100
x1_mod_p: 1110010000, x2_mod_p: 11100100

HMAC TAG: 11100101111
```

Working Flow:

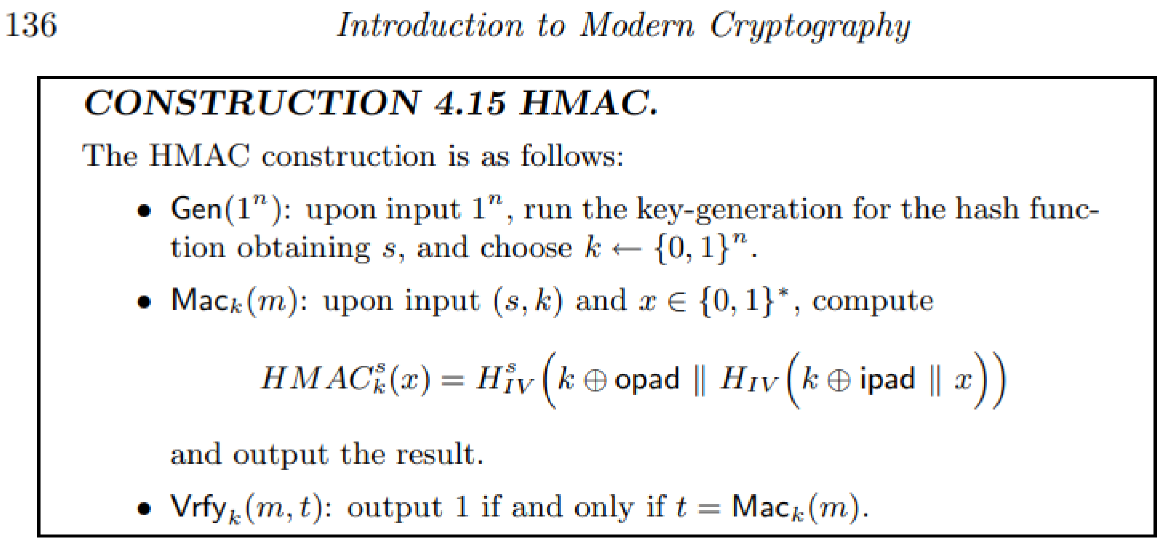
Refer to the below picture for better visualization. Here hs is our previously built DLP based hash function.

1. Upon getting the input, the function `construct_hmac(prime, generator, h, prime_bin, key, data, IV)` will be called.
2. At first, IPAD and OPAD two constants will be adjusted according to the length of the prime.
3. Then, data is padded with zero if necessary.
4. Then, x1 = k xor IPAD, x2 = initial vector. It is passed through DLP based hash func.
5. Then, a loop is run for d times, where d is no of data blocks.
6. For each iteration, x1 = corresponding msg block, x2 = previous func output.
7. After loop finishes, in the next iteration, x1 = length of data, x2 = previous hash func output, passed into DLP based hash func.
8. In the next iteration, x1 = k xor OPAD, x2 = initial vector. It is passed through DLP based hash func.
9. In the next iteration x1 = output from 7, x2 = output from 8, It is passed through DLP based hash func.

10. The HMAC TAG is the output of 9.



HMAC Construction



136 *Introduction to Modern Cryptography*

CONSTRUCTION 4.15 HMAC.

The HMAC construction is as follows:

- $\mathsf{Gen}(1^n)$: upon input 1^n, run the key-generation for the hash function obtaining s, and choose $k \leftarrow \{0,1\}^n$.
- $\mathsf{Mac}_k(m)$: upon input (s,k) and $x \in \{0,1\}^*$, compute

$$HMAC^s_k(x) = H^s_{IV}\Big(k \oplus \mathsf{opad} \,\|\, H_{IV}\big(k \oplus \mathsf{ipad} \,\|\, x\big)\Big)$$

  and output the result.
- $\mathsf{Vrfy}_k(m,t)$: output 1 if and only if $t = \mathsf{Mac}_k(m)$.