Merkle-Damgard Transform to obtain a Provably Secure Collision Resistant Hash Function

Assumptions: Used previously built PRG to generate h and DLP based hash function here.

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, it'll ask to input the data whose max length can be $2^{l(n)}$-1, where l(n) is length of the prime in bits.

Output Format:

1. It will output the hashed data after merkle damgard transform.

```
PS C:\Users\Sudipta Halder\Desktop\IIITH ASSIGNMENTS\POIS> python .\7_merkle_damgard_transform_hashing.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-1)
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
Input a seed in binary for PRG: 1101010111

Randomly selected h from 1 to prime: 481

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

Enter data in binary(maximum length = 2047): 101010000010100101011101011010

Data after zero padding: 101010000010100101011101011010000
['10101000001', '01001010111', '01011010000']

The initial vector is: 00000000000
Now, this zi and xi will act as (x1, x2) in the dlp based hash function

Round #1
Vector z0: 00000000000
x1: 10101000001
Concatenated data inseted into dlp hash: 00000000000||10101000001
x1_mod_p: 0, x2_mod_p: 10101000001
Obtained hash for this round: 10011111001
```

```
Round #2
Vector z1: 10011111001
x2: 01001010111
Concatenated data inseted into dlp hash: 10011111001||01001010111
x1_mod_p: 10011111001, x2_mod_p: 1001010111
Obtained hash for this round: 00000000011

Round #3
Vector z2: 00000000011
x3: 01011010000
Concatenated data inseted into dlp hash: 00000000011||01011010000
x1_mod_p: 11, x2_mod_p: 1011010000
Obtained hash for this round: 10100110000

Last Round
hash (z0||length of msg)
z: 10100110000
L: 00000011110
Concatenated data inseted into dlp hash: 10100110000||00000011110
x1_mod_p: 10100110000, x2_mod_p: 11110
Obtained hash for this round: 10010001010

Hashed data after merkle damgard transform: 10010001010
```

Working Flow:

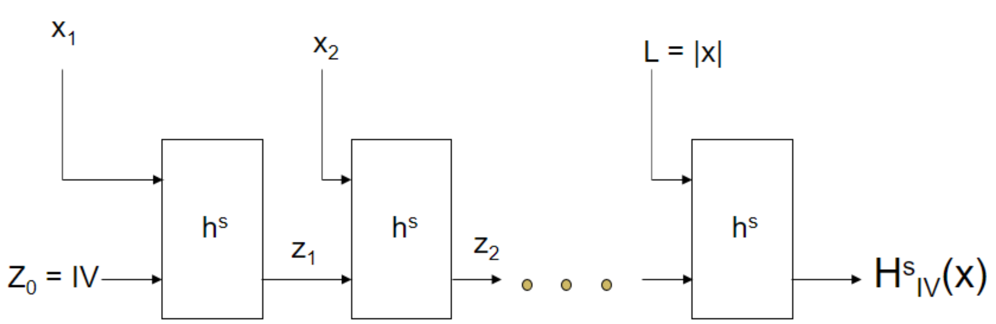
1. Upon getting the input, the function merkle_damgard_transform(prime, generator, h, prime_bin, prime_bin_len, data)` will be called.
2. For next steps clarification, refer to the below 2 diagrams.
3. Here h_s refers to our previously built hash function.
4. So, first if the data length is not multiple of the length of the prime, then zero is padded at the end to make it so.
5. Next, the data is divided into blocks of block-size = prime length.
6. Then, an initial vector is chosen whose length = prime length and all 0’s(00000..).
7. Then a for loop runs d times, where d = no of data blocks.
8. In first iteration, x1 = initial vector with all 0’s, x2 = first block of data. These two go as input to DLP based fixed length hash function. The output goes as x1 for next iteration.
9. For next iteration onwards, x1 = previous iteration hash function output, x2 = corresponding data block.
10. For last block, x1 = output of the hash function in previous iteration, x2 = length of the data.
11. The output of the last block is the merkle damgard transformed hash data.

CONSTRUCTION 4.11 The Merkle-Damgård Transform.

Let (Gen_h, h) be a fixed-length hash function with input length $2\ell(n)$ and output length $\ell(n)$. Construct a variable-length hash function (Gen, H) as follows:

- $\mathsf{Gen}(1^n)$: upon input 1^n, run the key-generation algorithm Gen_h of the fixed-length hash function and output the key. That is, output $s \leftarrow \mathsf{Gen}_h$.
- $H^s(x)$: Upon input key s and message $x \in \{0,1\}^*$ of length at most $2^{\ell(n)} - 1$, compute as follows:
  1. Let $L = |x|$ (the length of x) and let $B = \lceil \frac{L}{\ell} \rceil$ (i.e., the number of blocks in x). Pad x with zeroes so that its length is an exact multiple of ℓ.
  2. Define $z_0 := 0^\ell$ and then for every $i = 1, \ldots, B$, compute $z_i := h^s(z_{i-1} \| x_i)$, where h^s is the given fixed-length hash function.
  3. Output $z = H^s(z_B \| L)$

Merkle Damgard Transform



Theorem: If (Gen,h) is a fixed length collision resistant hash function, then (Gen, H) is a collision resistant hash function