Collision resistant hash function using Merkle Damgard Transform

Sarthak Mahajan, 2018111005

March 7, 2022

1 Code structure:

- This folder contains merkle.py and run.py code files
- merkle.py contains the code to create the output of a collision resistant hash function
- run.py is used to take inputs and return outputs of the collision resistant hash function

2 Instructions to run the code:

- Run the following command in the folder where this document is contained: python3 run.py
- There will be prompts asking you to input the relevant information serially, and will return the output.

3 Explanation of Functions in run.py:

3.1 run():

- It asks for inputs of key length in binary, the value that needs to be hashed to length n, where n= key length (all in decimal format)
- Calculates q: the order of the group for DLP ,g: the generator of this group, and h: a random element in the group
- prints the hashed value

4 Explanation of Functions in merkle.py:

4.1 merkle(m, key_len, q,g,h,iv=-1):

- q,g,h are in decimal format.They are needed to be passes to the fixed_hash function. p is the largest prime possible in n bits, where n= length of the key, g ,h are random numbers in the range(1,q-1); note that they are fixed for the program in terms of q(so that same seed gives same value for different iterations of the program), but they could be any random number in the given range.
- All inputs are in decimal format, m is the message, key_len is length of the key in binary format, iv is the initialization vector
- It returns a binary number representing the hash of the number in n bits where n= key_len
- First we convert m into binary format, and cut it to chunks of size key_len, if the last chunk's size is lesser the key's size we pad it with zeros on the right side
- We use the Merkle Damgard transform using a for loop iterating over message chunks.
- Uses fixed_hash, pad functions.