Soumodipta Bose| 2021201086

Using Collision resistant hash function to build H-MACs(Code)

HMAC:

- **hmac(k, iv, message):**Calculates tag for HMAC

  Working:

  - calculates xor of ipad and opad with key and performs hash with iv for first and semi last stage
  - In between uses merkle damgard to calculate hash of the entire message using the ipad hash as input

  Args:

  k (binary string n bits): key for hmac

  iv (binary string n bits): Initialization vector

  message (binary string): message whose tag will be calculated

  Returns:

  binary string (n bit): Tag returned

- **hmac_verify(k, iv, message, tag):**

Verification algorithm for hmac

  Args:

  k (binary string n bits): key for hmac

  iv (binary string n bits): Initialization vector

  message (binary string): message whose tag will be calculated

  tag (binary string n bits): Tag send to reciever for verification

  Returns:

  boolean: True if tag verified else False

Usage:

1. The Gen over here is **get_group_parameters()**
2. Fix a binary string as Initialization Vector for Merkle damgard transform, here we are using getrandbits to generate a random initialization vector.
3. Take an input message like "Hello world" on which we will find our HMAC.
4. Then take a key as input preferably of the same size as iv or less than equal to 16bits, as it supports upto 16bits right now.

5. Convert the message into binary using utility function **str_to_bin(s)**
6. Use **hmac()** to calculate tag of the binary message.
7. Use **hmac_verify()**to verify the tag generated and make sure that no corruption has occurred in the message.
8. Since prime numbers are fixed the limitation is that it can only handle upto 16bits, will be expanded later in future revisions.

Crypto Library:

- **hash(x1, x2):** Generates fixed length hash using DLP

  Args:

  x1 (int): input to be compressed

  x2 (int): input to be compressed

  Returns:

  int : integer after 50% compression

- **generator(p, q):**Returns a primitive root of p

  Args:

  p (int): safe prime number

  q (int): safe prime number

  Returns:

  int: primitive root

- **get_group_parameters():**Gets the group parameters

  Working:

  For now prime no. selection is static using a 16 bit Sophie Germain safe prime, will move towards safe prime generation in next update with more time

  Returns:

  p,q,g,h: Returns all the group parameters

- **hash_wrapper(x1, x2):**hash wrapper for binary strings

  Args:

  x1 (binary string): binary number

  x2 (binary string): binary number

  Returns:

  binary string: binary number

- **merkle_damgard(iv, message):**Merkle Damgard Transform

Working:

- Message length = L len(iv)=l
- Makes message a multiple of length l and appends message size
- iterates and applies Fixed length hashing using x2 as message and x1 as hash of previous iter

Args:

iv (binary string nbit): initialization vector

message (binary string): message in binary

Returns:

binary string nbit: Hashed value

Utility functions:

- **dec_to_bin_wo_pad(x):** Converts decimal to binary without padding
- **dec_to_bin(x, size):**Converts decimal to binary with padding
- **bin_to_dec(x):** converts binary to decimal
- **str_to_bin(s):** converts string to binary
- **repeat_string(inp, length):** Creates repeating sequences of input string of size length
- **xor(bin_x, bin_y):** xor of the two binary strings