DS-PROJECT 2024

JANUARY 9

IUST
Authored by: ALI ZALI
401400008

HASHING:

Allowed hashing: MD5, SHA-1, SHA-2, NTLM, LANMAN

Now we want to compare above hashing functions in **cryptographic purposes**,

> **MD5**: This is one of the oldest and most widely used hashing algorithms. It produces a 128-bit hash value and is no longer considered secure for cryptographic purposes due to its vulnerability to collision attacks 1.
>
> **SHA-1**: This is another widely used hashing algorithm that produces a 160-bit hash value. Like MD5, it is no longer considered secure for cryptographic purposes due to its vulnerability to collision attacks 1.
>
> **SHA-2**: This is a family of hashing algorithms that includes SHA-224, SHA-256, SHA-384, and SHA-512. These algorithms produce hash values of 224, 256, 384, and 512 bits, respectively. SHA-2 is currently considered secure for cryptographic purposes .
>
> **NTLM:** This is a proprietary hashing algorithm developed by Microsoft. It is used for password storage and authentication in Windows environments. NTLM is no longer considered secure and has been replaced by Kerberos 2.
>
> **LANMAN**: This is an old hashing algorithm used for password storage in Windows environments. It is no longer considered secure and has been replaced by NTLM 3.

It depends on the specific use case. **MD5** and **SHA-1** are no longer considered secure for cryptographic purposes due to their vulnerability to collision attacks. **SHA-2** is currently considered secure for cryptographic purposes.

However, it's important to note that the choice of hashing algorithm depends on the specific use case. For example, **NTLM** is used for password storage and authentication in Windows environments, while **LANMAN** is an old hashing algorithm used for password storage in Windows environments.

In general, it's best to use a hashing algorithm that is currently considered secure for cryptographic purposes, such as **SHA-2**. However, the choice of hashing algorithm depends on the specific use case and the requirements of the system.

SHA-2

Step 1 — Pre-Processing

Convert text to binary,

Append a single 1,

Pad with 0's until data is a multiple of 512, less 64 bits (448 bits in our case),

Append 64 bits to the end, where the 64 bits are a big-endian integer representing the length of the original input in binary. In our case, 88, or in binary, "1011000".

Now we have our input, which will always be evenly divisible by 512.

Step 2 — Initialize Hash Values (h)

Now we create 8 hash values. These are hard-coded constants that represent the first 32 bits of the fractional parts of the square roots of the first 8 primes: 2, 3, 5, 7, 11, 13, 17, 19

step 3 — Initialize Round Constants (k)

Similar to step 2, we are creating some constants. This time, there are 64 of them. Each value (0–63) is the first 32 bits of the fractional parts of the cube roots of the first 64 primes (2–311).

Step 4 — Chunk Loop

The following steps will happen for each 512-bit "chunk" of data from our input. In our case, because "hello world" is so short, we only have one chunk. At each iteration of the loop, we will be mutating the hash values h0-h7, which will be the final output.

Step 5 — Create Message Schedule (w)

Copy the input data from step 1 into a new array where each entry is a 32-bit word,

Add 48 more words initialized to zero, such that we have an array w[0…63],

Modify the zero-ed indexes at the end of the array using the following algorithm:

For i from w[16…63]:

s0 = (w[i-15] rightrotate 7) xor (w[i-15] rightrotate 18) xor (w[i-15] right shift 3)

s1 = (w[i- 2] rightrotate 17) xor (w[i- 2] rightrotate 19) xor (w[i- 2] right shift 10)

w[i] = w[i-16] + s0 + w[i-7] + s1

Step 6 — Compression

Initialize variables a, b, c, d, e, f, g, h and set them equal to the current hash values respectively. h0, h1, h2, h3, h4, h5, h6, h7

Run the compression loop. The compression loop will mutate the values of a…h. The compression loop is as follows:

```
for i from 0 to 63
S1 = (e rightrotate 6) xor (e rightrotate 11) xor (e rightrotate 25)
ch = (e and f) xor ((not e) and g)
temp1 = h + S1 + ch + k[i] + w[i]
S0 = (a rightrotate 2) xor (a rightrotate 13) xor (a rightrotate 22)
maj = (a and b) xor (a and c) xor (b and c)
temp2 := S0 + maj
h = g
g = f
e = d + temp1
d = c
c = b
b = a
a = temp1 + temp2
```

Step 7 — Modify Final Values

After the compression loop, but still, within the chunk loop, we modify the hash values by adding their respective variables to them, a-h. As usual, all addition is modulo 2^{32}.

Step 8 — Concatenate Final Hash

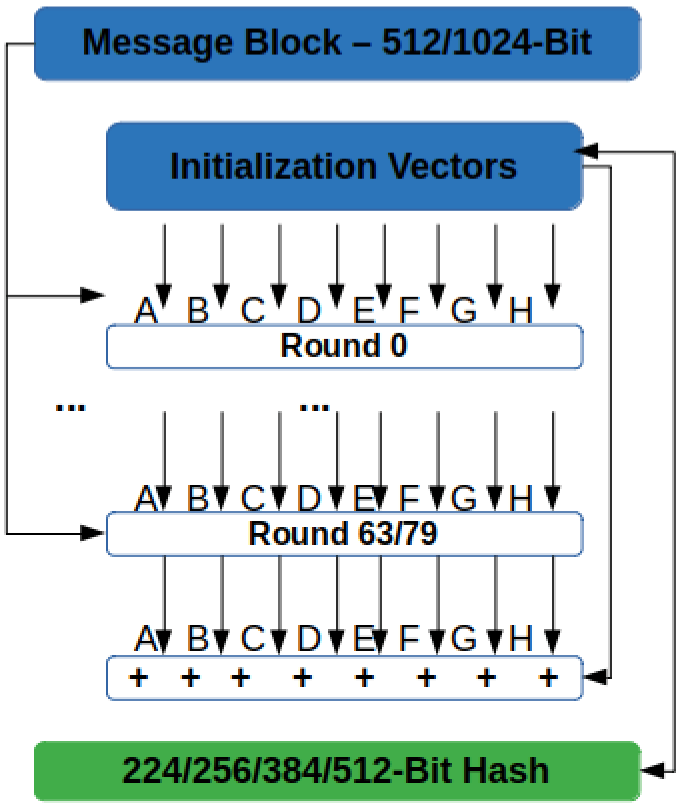
Ex:

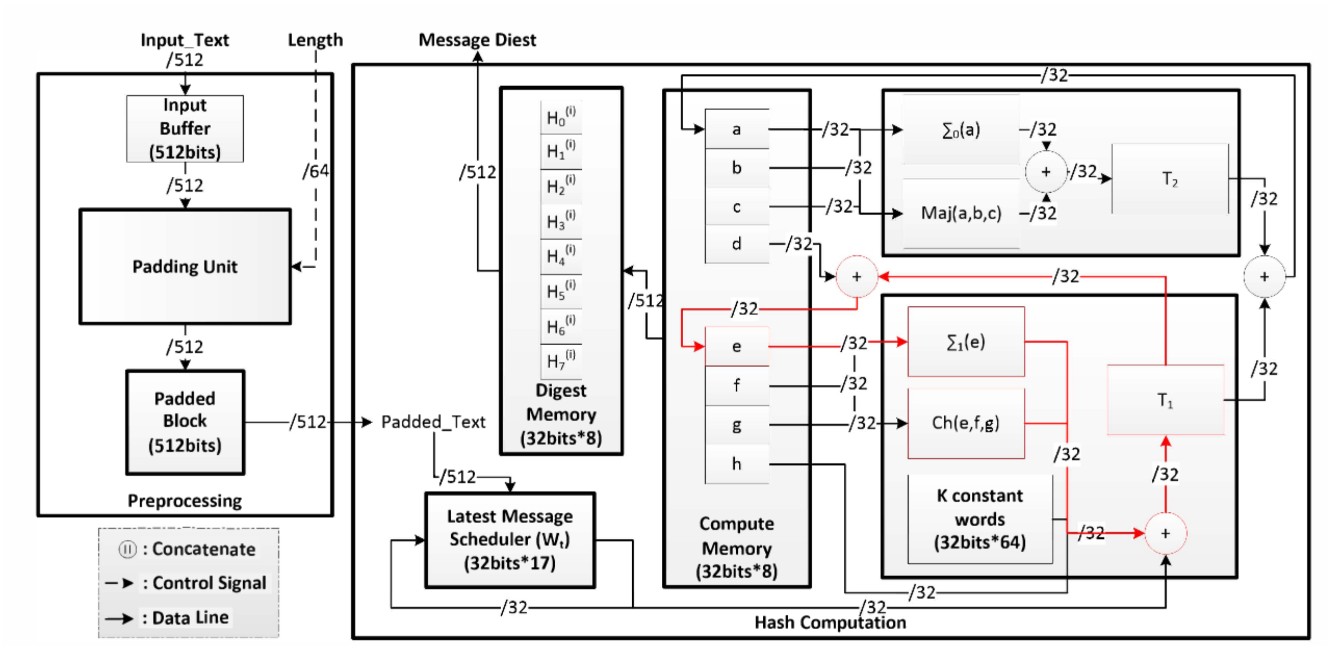
Data : Geeksforgeeks

SHA2(256):
86d755349c6b9f95f365c6ffe7734f25bf2b00cabe8c6bc5f2b8b746c1aac332

This algorithm in order O(n),

SHA-2 Process Overview





Methods and Functions:

Hash Function:

(sha_256(name)):

Get the string as input and convert to cryptographic password in order

O(n)

Is child:

Is_child(other)

It’s get self and other as input and checks that the other

Is a child of self or not

Output is True & False

O(n)

Is siblings:

Is_siblings(other)

Checks that two inputs have a same father or not

Output is True & False

O(1)

Farest child:

farest()

find the farest child of his self

out put is a name but in hash way

o(n)

farest relationship:

max_width_and_values(root)

its look like bfs but with some changes and calculate the width of

each level and compare with others

output is max width, max width values

O(V+E)

here V is number of vertices and E is number of edges in Family tree

far relation:

far_relation(other)

find a same grand fhater of two inputs

output is grand father name

O(n)

Is related:

is_related(other)

checks that two person has a relation or no

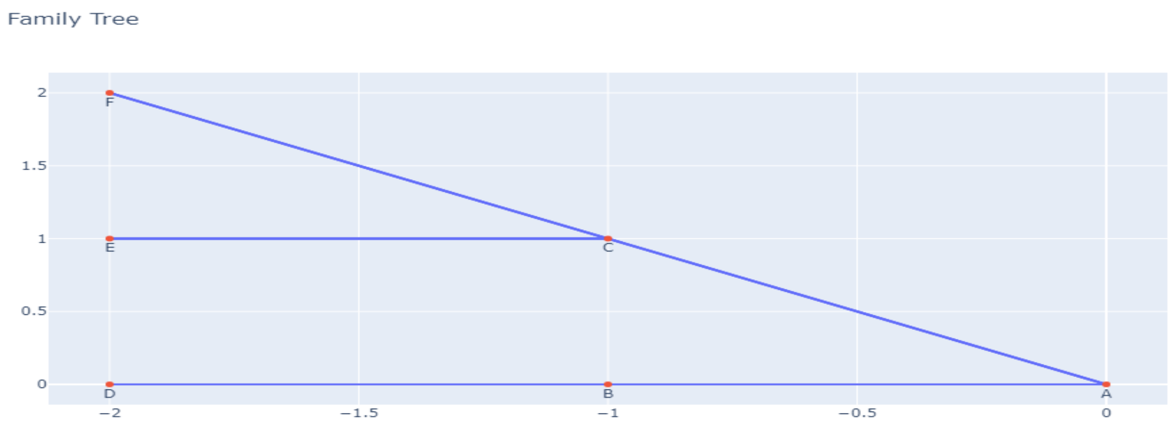
it’s mostly same as far relation but difference is in output length

O(n^2)

Its calculate two times of function find_ancestors

Visualize:

The provided function, visualize_family_tree, generates a visual representation of a family tree using the Plotly library. It employs a recursive traversal approach to map out the family relationships and position the family members in a tree structure. Family members are depicted as nodes, and their connections are represented as edges in the tree diagram. The function then utilizes Plotly’s Scatter plot to render an interactive and clear visualization of the family tree. This optimized function offers a streamlined and effective solution for visualizing complex family structures.


Family Tree
F
E
C
D
B
A