ECRYP PROJECT

SHA-256 HASH FUNCTION

Jan 19, 2022

AUTHORS

Name	**Student Number**
Gaurav Chauhan	309602
Laksman Sivaram Senthilkumar	317289

DESCRIPTION

SHA-2 (Secure Hash Algorithm 2), of which SHA-256 is a part, is one of the most popular hash algorithms around. SHA-2 is known for its security and its speed. In cases where keys are not generated, such as mining Bitcoin, a fast hash algorithm like SHA-2 often has the upper hand. SHA-256 is used for authenticating Debian software packages and in the DKIM message signing standard.

ALGORITHM

1. **Pre-Processing**
    - Convert given input to binary.
    - Append a single 1.
    - Pad with 0’s until data is a multiple of 512, less 64 bits.
    - Append 64 bits to the end, where the 64 bits are a big-endian integer representing the length of the original input in binary.

2. Initialize Hash Values(h) and Round Constants(k)

- 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.
- Next, we create 64 constants. The values of the constants are the first 32 bits of the fractional parts of the cube roots of the first 64 primes.

3. Chunk Loop

- The following steps will happen for each 512-bit **chunk** of data from our input.
- At each iteration of the loop, we will be mutating the hash values **h0** to **h7**, which will be the final output.

4. Message Scheduling

- 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 of length 64 words.
- Modify the zero-ed indexes at the end of the array using the following algorithm:

```
for i from 16 to 63
    s0 := (w[i-15] rightrotate  7) xor (w[i-15] rightrotate 18) xor (w[i-15] rightshift  3)
    s1 := (w[i-2] rightrotate 17) xor (w[i-2] rightrotate 19) xor (w[i-2] rightshift 10)
    w[i] := w[i-16] + s0 + w[i-7] + s1
```

5. 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** to **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
    f := e
    e := d + temp1
    d := c
    c := b
    b := a
    a := temp1 + temp2
```

- All additions are calculated modulo 2^32.

6. 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** to **h**. As usual, all addition is modulo 2^32.

7. Concatenate Final Hash

- The final hash(digest) is produced by a simple string concatenation.

```
digest := hash := h0 append h1 append h2 append h3 append h4 append h5 append h6 append h7
```

PROJECT STRUCTURE

The SHA-256 implementation is written in Python3 with reference to the [FIPS 180-4] Secure Hash Standard Specification, published by the **National Institute of Standards and Technology(NIST)**, on August 15.

Functions such as **Ch(x, y, z)**(choose), **Maj(x, y, z)**(majority), **Sigma0(x)**, **Sigma1(x)**, **Gamma0(x)**, **Gamma1(x)** and **ROTR**(rotate right) are defined as given in the specification.

$$\begin{aligned} Ch(x,y,z) &= (x \wedge y) \oplus (\neg x \wedge z) \\ Maj(x,y,z) &= (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) \end{aligned}$$

$$\begin{aligned} \sum_0^{\{512\}}(x) &= ROTR^{28}(x) \oplus ROTR^{34}(x) \oplus ROTR^{39}(x) \\ \sum_1^{\{512\}}(x) &= ROTR^{14}(x) \oplus ROTR^{18}(x) \oplus ROTR^{41}(x) \\ \sigma_0^{\{512\}}(x) &= ROTR^{1}(x) \oplus ROTR^{8}(x) \oplus SHR^{7}(x) \\ \sigma_1^{\{512\}}(x) &= ROTR^{19}(x) \oplus ROTR^{61}(x) \oplus SHR^{6}(x) \end{aligned}$$

$$ROTR^n(x) = (x >> n) \vee (x << w - n)$$

The **round constants** and **hash values** are defined as arrays. The main hash function, **sha256_hash(message)**, takes the input message which should be a string object and then computes the hash and returns the digest, which is also a string object.

SHA-256 is a cryptographic (**one-way**) hash function, **so there is no direct way to decode it**. The entire purpose of a cryptographic hash function is that we can't undo it.

One method to find the original input message is by **brute-forcing**, where we **guess** what was hashed, then hash it with the same function and see if it matches. Unless the hashed data is very easy to guess, it could take a **long time**. This approach to decode a digest is implemented in **sha256_decode(digest)** function, which takes the digest which should be string object and returns the original input message if possible with the given computational power.

Our test bench could only find the original message, which at max can be of length 3, in reasonable time. Also, our implementation can only find the original messages, which only contain ASCII printable characters, because it generates all possible permutations of input message strings of all possible lengths with the ASCII printable characters.

Testing is done by **asserting correct hashes with reference test vectors** taken from https://www.di-mgt.com.au/sha_testvectors.html, https://www.cosic.esat.kuleuven.be/nessie/testvectors/hash/sha/Sha-2-256.unverified.test-vectors and https://csrc.nist.gov/projects/cryptographic-algorithm-validation-program. It is implemented in **tests.py**. Our test bench takes on average 7 seconds to complete the testing. Extended tests, which include test vectors from CAVP, are also available to test.

The main user interface is a **shell** which has a few simple commands relevant to SHA-256 hash computation, such as hash generation and fining the original message of a given hash. Tests and extended tests can be invoked in the shell too.

OUTPUT

Shell

```
PS C:\Users\xdpx\Desktop\exryp> & "C:/Program Files/Python310/python.exe" c:/Users/xdpx/Desktop/exryp/shell.py
SHA-256 Hash Computation Shell (Type "help" for commands)


>>> help

hash [input message]  : generates the SHA-256 hash of the given input message string
hashx [input message] : generates the SHA-256 hash of the given hex
find [hash]           : tries to find the original input message of the given SHA-256 hash by brute-forcing
tests                 : runs the tests
extests               : runs the extended tests
exit                  : exits the shell

>>> hash abc

Hash : ba7816bf8f01cfea414140de5dae2223b00361a396177a9cb410ff61f20015ad

--- 0.0010166168212890625 seconds ---
>>> hashx deadbeef

Hash : 5f78c33274e43fa9de5659265c1d917e25c03722dcb0b8d27db8d5feaa813953

--- 0.0009989738464355469 seconds ---
>>> hash b9

Hash : cb440fe2f7ec20d54f4726630cebadb8673965ccb57a64bbeda757842fd26375

--- 0.001012563705444336 seconds ---
>>> hash cb440fe2f7ec20d54f4726630cebadb8673965ccb57a64bbeda757842fd26375

Hash : f5671a3a6cb987cacdcf97d95b4214c25feef1a80f50ca7bf3a97092dd17085c

--- 0.00201416015625 seconds ---
>>> find cb440fe2f7ec20d54f4726630cebadb8673965ccb57a64bbeda757842fd26375

Original input message : b9

>>> find feedbed

Original input message : Invalid SHA-256 hash

>>> exit
PS C:\Users\xdpx\Desktop\exryp>
```

Tests

```
>>> tests
Input string          :  ""
Expected SHA-256 hash  :  e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
Generated SHA-256 hash :  e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855

Input string          :  "a"
Expected SHA-256 hash  :  ca978112ca1bbdcafac231b39a23dc4da786eff8147c4e72b9807785afee48bb
Generated SHA-256 hash :  ca978112ca1bbdcafac231b39a23dc4da786eff8147c4e72b9807785afee48bb

Input string          :  "message digest"
Expected SHA-256 hash  :  f7846f55cf23e14eebeab5b4e1550cad5b509e3348fbc4efa3a1413d393cb650
Generated SHA-256 hash :  f7846f55cf23e14eebeab5b4e1550cad5b509e3348fbc4efa3a1413d393cb650

Input string          :  "abc"
Expected SHA-256 hash  :  ba7816bf8f01cfea414140de5dae2223b00361a396177a9cb410ff61f20015ad
Generated SHA-256 hash :  ba7816bf8f01cfea414140de5dae2223b00361a396177a9cb410ff61f20015ad

Input string          :  "1234567890" * 8
Expected SHA-256 hash  :  f371bc4a311f2b009eef952dd83ca80e2b60026c8e935592d0f9c308453c813e
Generated SHA-256 hash :  f371bc4a311f2b009eef952dd83ca80e2b60026c8e935592d0f9c308453c813e

Input string          :  "abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq"
Expected SHA-256 hash  :  248d6a61d20638b8e5c026930c3e6039a33ce45964ff2167f6ecedd419db06c1
Generated SHA-256 hash :  248d6a61d20638b8e5c026930c3e6039a33ce45964ff2167f6ecedd419db06c1

Input string          :  "abcdefghbcdefghicdefghijdefghijkefghijklfghijklmghijklmnhijklmnoijklmnopjklmnopqklmnopqrlmnopqrsmnopqrstnopqrstu"
Expected SHA-256 hash  :  cf5b16a778af8380036ce59e7b0492370b249b11e8f07a51afac45037afee9d1
Generated SHA-256 hash :  cf5b16a778af8380036ce59e7b0492370b249b11e8f07a51afac45037afee9d1

Input string          :  "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789
Expected SHA-256 hash  :  db4bfcbd4da0cd85a60c3c37d3fbd8805c77f15fc6b1fdfe614ee0a7c8fdb4c0
Generated SHA-256 hash :  db4bfcbd4da0cd85a60c3c37d3fbd8805c77f15fc6b1fdfe614ee0a7c8fdb4c0

Input string          :  "a" * 1000000
Expected SHA-256 hash  :  cdc76e5c9914fb9281a1c7e284d73e67f1809a48a497200e046d39ccc7112cd0
Generated SHA-256 hash :  cdc76e5c9914fb9281a1c7e284d73e67f1809a48a497200e046d39ccc7112cd0


All tests passed successfully
--- 7.1955835819244385 seconds ---
>>>
```

Extended Tests

```
de594c3b2e4fb9f2996e0418ec8698a4c087c14a2687717f97e228e75afe295caae2f16513f47a45b4124a7c5ebacbacc562951233bf89f43ff85b703ec77f168c2278fbe6e57a0e7192125f4642d73f227d806287081bd30149b9d44fdb9002966
7622f9925b7826bd0343bc537c666e660f174b44786e1bb8846c3edcb639ebd213a4695f9cb471e188db7a859fcf3abae49569e676dec857b897627cb0bc1155ad6d45282d430176fde4262da2d5f41ff890ceb319d73ddo804738456f30a3d68da41
554d4cede62oa8549b24e211e76768e6b17379f842a24a449a0ba3ea73cfc72624b5afd118fd7e76a7c6b5bbfa7a6b6c97b97dea52decd51cf35a8e277140ffb2748777a1e3cc3211f3c12be099d0316f45023da6cd200339a718c72a5ca172983922
e59648d08dc67f173788363c26e5df406391f107552925ba91b9e569f38101f5eef9a52d2012883720abf6532beb4af19fa6d81eaf473d40896dbf4deac0f35c63bd1e129147c76e7aa8d0ef921631f55a7436411079f1bcc7b98714ac2c13b5e7326e
60d918db1f05ffb19da767a95bb141a84c4b73664ccebf844f3601f7c853f009b21becba11af3106f1de5827b14e9facb84b2cbf16d18c045622acb260024768e8acc4c0ae2c0bd5f60a98023828cdec18ed8dc298a306c38d1ece01509f3265b5f8cb
f441f0525097e8b48234bf69f65cf402c7540a023ed231ef95b222c900ea0bfaecc02c6d8b3b81648ad7a165237cc6b557b1ce287b0ea137f4ef54534070ee793695a9078ec89bcea38995687861 4ccbf917b61f8427b7cda870fdd92d2d297154262
fc65f28ff1a54b2651afff12d6f36ea8c90610 7bbda399ce5e2cf0a430ad0dd8652084175726bad725bf1593c7959f16221894f5852ddad3172fef866b3321755491fd44fba009b42ec0b6c4fb9e901d7eb3b8acf70e94911f54c538bd0559c57400
42b6df4a07c3e00bba0934d92a684b39592a576331e5a44672a227cce f3e595ffa1146ac1dcee0a70baa9acfd5c132b361b5ceb519984b0ee00cd2124aa8acb50c9e574fb19bd99c8fef5487faeedb28b796848bbb372beb3f5bde55ed2cb140b60a53
bba2df471f330208b09ffb8eda04315a06d693aa53d9bf8939ef6f3a68de6e1975f79f50b3d484665e4ee71124ed794be3a2baa7b5b918e62a095bc5d46e401a0979641fe465640e8d4d43eeba9d0cac76c7b86d22375123b988585e58f86566fd19
0d868eca08aa1e66932d6d3b14ecad3efd9f8cfcf2696ed42eadfa642324d9416d2cbaebb8639a00a17542afda32117051e4fbf243dfd255a559c49ac37c265827ba70b0bc618882336f43e1a6a729c57be478008cae6c74840bbe828c976ac628d7b
6015bcb705612c277bac0727da645480e0e14fdc497956aef05c89d30f22c2c96c6dfc9dae30617e6206fbd957975bbba0524f563289e1f5f09bdb6fd46fa6117e78e854f91d71699fcfcadfaa7d4db8fcb04bed08d68d11677b5085b295c1d414cb1
2456c84c669737af6c33992a5a9149fc7f9330bb291d38f6bed10318081dde8fd178f02eb0e8b7d022c8b63fdcc867546035775fcf7b32c8fee83df7cbb28372b23c71459b9566a7f64165da0a3d0e538a3dcc1b6a384f75f0263dc10e0924a0ef2ab
459d0a52b7c112710c58cf724422533396b8a25d7644be166c3e7828aa62b1ca1f32f620ed969b021ec609fe926958a03cff21f08f7c8d3d3235b219fb0020a51b97b60f963ebb58f7a62a5b41104c0b28b58cfc81668825f87064e401c263421152b8
790dbc99b3032c9615187f29fcc1a58e86364ad45524b5358fa2f0a3296729a3663a585e9aa922f534fefd16fb6f96cd98957095c5520cdcd24c8d107e387e520de855a3296544ef1c1ddd43b919a4ff13986 1f06ae5280d5aa5aaeb8f7d74ed6ea560
93c2e697a30c29c4ac145aa99a372f1a03ae72495f52a40cfddedc12b6e9115aea5ea516c5a4223a8d0a0073c8b4abe3c6188fdd6d4ab627c9f4eab468fdc2a91945274ed18465a368f291a0050c9d638a31944091b35a8fd26a1ff65e2d17dfa32ef
3ac412d8293b276849ad9af71fdf272363f771d0fa99996e24510e7bf731a7480cbbefff7801c0e5fd0a13dd8278162ec1687f85409a203e82d2bcdf7e7d1ae5509857c42fce80299fe06182e74a97c0c624ed5b624e59781af9407fb28b34f7024f
42d36eb92bb95f72cee379ed363daf2625b48e60d8489b23dfa57789c0dd2276b4575a01c2349171d2a58bcf29e659b868cdac1c30a02a160c078b6faa7e0696711d43447ea2108db3d34ec1bf9cfe802f6012d335445a4624829f8a600b18e9b3c
f13a9787910f2fb27676fd809e7ea1a34c7306e766b2e7ae1bbb919cc888ea931d1eb2e27c6109b9a12c31e188a196a98bbe0b24cc315791d26ef01b77fe06c3011ac39a8f78d233b765 1e586d14dcfc2636cb713ecabadb97374ce58498f8b2e5575
31793fd9207fe484a4e147f7b826502cd378525 1973b23e2b62b7fdc74a10fce9c04f97511dbffe3f2c46887c25904b99df69e97b416bac18fadad67b71cc320eff8def185d41ae8558cbdae6ccee38b8cfb2bfe92d0aa99815b3ca1d115f21493b13
adeeafce81a023c6b1bc15fc8f2b171284e6a1fd65c351b0c82b31112f022ddaa78dcfbac9f203eeef415c566e00c2c933f06ff18ee7674aba548592dc8214b1af8e929242f87c81b0cebe8106b5267ba39c5b51987e38858dce1d1f8d0cfee2bd61d2
17e5a5d41bb0c4aaf0e7b0a8c66e5b0291e4d05bfddcf886 1bb31b32ea5ba80cb02472c11969b3b02a7f7bc025feace44726b6382012544f1bd1256744f4b1b0ff81f7b9f7462c5c92507f1316df228ec5c0786378b871e69479c3e26f232f5d6a709
d3551d08f0ecced52f8158a2c40a234af448449c1cb1a1f6f5ae56171606582ebb9a5836c454eb86015ae7a4ac87105b371bf40d49b1134a037243a0878953b5bbd6ef944ae7c345ec24e4a0e8496b62d71a6381aa52e5bdeedc81784f45e0c75b72a
8c9898ea0387a47153d7e3a7c895aab58a1497a5e794052d7457624478c24a44c7e8932c887322b422478418af64a389c152d12c7a6803e0fb0050dcf2b9d65a35a53b9845b9c3835fddd45dfd12e28f8845e03686b3707ef6003e7c1cd4f8d7406ee
0d1cdc41d7b56fb630c1438fe33196e53389f1ec1540fe789c6599c0b589296214d831a86e89220ae97974f4d112f4c98c726027d0c9316d1303b87a43a86cb8b800835a677abfe1584e8be55a624612f56bdf71a054a2e834e35105a19a77f7dfdbf
9dd2850ee446658ab0eae6e833c855bb9650eda7f8f4e74d8de73526f12773b2bcbb1bd35639f8730d8cdd6d64f496abae4e1f8cdc961488940a691683515bcdf37ba6caa0cbf953c752a7b9819e9f834ff39ec8f6d8a3dd8dd5a431d47c7f74c7a633
ff73ff507009c5ac9431cc588ba0c6d226edc17c94a0f14d3e8db0c7ef60c3293878dfe513f96b54c61c88a90aca4f246d6a5988f5f785ce0655f51b85e55af03e5772a083bfcf0816ebd97a4af416fa6414a9ad47b7198e51d55463807ef4f0d9b7c
06a0a84762e4e46c8b39147a4bdd594b8d4d40b36f5e6b4d48726551890d040d229ee70ea3034d45b3c28eb80d686918fe6e219636b8f9b7e6fc08f4e3bed9bafc778aab274913e9cfd570732ab3fb434c9ba09285812325804955 1e56f6705f2af0
5b56642c2b93df65c443a6caa5b167a4040d2438206d2cefd3114ab466eb3c9eaa5e66cf4447c89c493a2eee0b0ea6e7329b37c90ec2d0142bae7fef265ae3c9c053e44031c0a142bf9faa728e5170cdba59fa8da361d94d887d5d6f58b409bbc4bd4
548990653a04dfb841fd784ac9cc4cfd34c88512de212074dfba30295badf22f1af2522c5fe1cd423bd8eae429d7a862bcd649ab61bf0d3b55daf4b6f0f390c503d7c1bdea453b5ef145bd8191802056bd9e0455a404b6afe5b25977f02f902cabo46
f988d91b2350ebe4091b5584d4f938a45803984a5291beadeeadda488dc7ed2dc4aae69ca8ae0bd4492f9b297c3fb257de986c1615d44dee59e1e14d34af9fd7852b13fdcb713dd1a03d341884a30ea1dc0104d63a31d291df035d317fea98ec44f5a
867150147831 72e667a748f162c5c26a8b34a0f133d89fb971bf6e0a01507efed010cc7f194b5e87a77d56a909d65efa0d5ccd6da9b5eb1d73422f97ffad8012af43a2905a98354b8362e9c459f0044336348eded53660d65a38a9efc42be13a66727
90496d875a67e0078dfdd8340dab8547be140ca9f88891b635e195c20daa8359658785cbe3d09ce8a580f009324e6550b0196e305889262f28f49dead77e6f5a0e859c57d53c935a4c9590879b6528eb2bc3230217b0897cddfeff405a6a54b2f50c5
8311af1ede4ea0660b73037f9a097d9d0271b45e325bec666cc7cb65ae780e361639838d10fe79907a0da0efef85d2420a84e905bb331167895269a9a88319d460f539586762ab172e4a7f305f7ae36cb88c96d91aada0b4ddd3418c670e27a5fdede3
9bd8659e477cbe08e645af927843dbdd67489b72693efeb3a7be0e121fdf5580474ca028f39a035e78d81dd212679d0a830c050ffd43af6642d60d410aaf34f7a5ea9cb2e12f21672e3f4e0c00ccdb05758e74df3893bd40a5d7921e2e149330fddbe
0a2dae4210d50a3caa60b1b9db685f7704ae2d7302b18e8261052b779139747f462a6610a37252b170afbfce905fb6f7fb8c2b6100ee231507f403fee88ba5561580d4de4cdf600bf9e9816c9da1e1d2b91a1d966d04cdb98d3be55fb77af2daeeed7
50b8b60b494accaa12441d372afb3d47e7395b9e0e867595a1a6c8bff8638bcb138ddcac2f3efbf89762b68ebd77247c89929620f1a3cb8dcaf9632fde0996b33e6b2621da25924b4e2c8d6bff28ae0867786919ad763e6d79fc304a062 77955795a7
cb17186fb6bdfa98a16189544b228f3bcd3698737ff55b6185799459b796a63c6a61cea9d20f1e296d62f474c43750b77944e5f1c09072f019dbeeb64e9bc8dec4605d8e0322cdd97f56cc43084f5c983e5848556543 66fd5659ea23c6c15e1d7da51
d82c683oa477b9f896563a5134c64e32814ea88b7f7af760f18bc91e656da92b72e98bc03f1c6bfb442830305529d681dc6bccae66da9b2e61b9c97e2397fdb92f7f6369b470529c570c2d3b329487981d148o462cdb992d792e34dd233e1c239657b
8da0d59b804566cf81ad5f0a7a0ccb3a8fbda673887c153d2e56c484f9230d752be52c1e35bc9af5a7446237fc072afef777665c264c18e6a3c059fde2e8368f9bb898f1cc8393d1bf18b1757219670275f8bbc7deb0248c68af929111e19737479bc
abab732d7e033aaeb277eac05e185e9e56b2450beaac784dd0308b7a5e8ca1f2fcd8852ddad9f7b7de264478e1891a391aa8996 4dae5ad8b7a829c2c9209db346ceb26c1b967cfac82ad574761443be3f0a910968239d23b11507ab978b3ce89e22b7
d7283736b9786544db4460f
Expected SHA-256 hash  :  33b6229592ca719e4e46f35b287617fedadd3b7c38be3c8c1c9f446d2d9085b3
Generated SHA-256 hash :  33b6229592ca719e4e46f35b287617fedadd3b7c38be3c8c1c9f446d2d9085b3


All tests passed successfully
--- 2.2315049171447754 seconds ---
>>>
```

REFERENCES

FIPS 180-4, Secure Hash Standard, Computer Security Resource Center Computer Security Resource Center, Information Technology Laboratory Information Technology Laboratory, National Institute of Standards and Technology, August 2015.

Cryptographic Algorithm Validation Program CAVP, Computer Security Resource Center
Computer Security Resource Center, Information Technology Laboratory Information Technology Laboratory, National Institute of Standards and Technology, August 2015.

Test vectors for SHA-1, SHA-2 and SHA-3, DI Management Services, June 6, 2020.

Project NESSIE, New European Schemes for Signature, Integrity, and Encryption*, June 23, 2005.

Andrew W. Appel, Verification of a Cryptographic Primitive: SHA-256, Princeton University.