# ICSS SHA-1

In cryptography, SHA-1 (Secure Hash Algorithm 1) is a hash function which takes an input and produces a 160-bit (20-byte) hash value known as a message digest – typically rendered as 40 hexadecimal digits. It was designed by the United States National Security Agency, and is a U.S. Federal Information Processing Standard.

SHA-1 produces a message digest based on principles similar to those used by Ronald L. Rivest of MIT in the design of the MD2, MD4 and MD5 message digest algorithms, but generates a larger hash value (160 bits vs. 128 bits).

SHA-1 forms part of several widely used security applications and protocols, including TLS and SSL, PGP, SSH, S/MIME, and IPsec. Those applications can also use MD5; both MD5 and SHA-1 are descended from MD4. Revision control systems such as Git, Mercurial, and Monotone use SHA-1, not for security, but to identify revisions and to ensure that the data has not changed due to accidental corruption.

These are examples of SHA-1 message digests in hexadecimal and in Base64 binary to ASCII text encoding.

SHA1("The quick brown fox jumps over the lazy dog")

Outputted hexadecimal: 2fd4e1c67a2d28fced849ee1bb76e7391b93eb12

Outputted Base64 binary to ASCII text encoding: L9ThxnotKPzthJ7hu3bnORuT6xI=

Pseudocode for the SHA-1 algorithm follows:

*Note 1: All variables are unsigned 32-bit quantities and wrap modulo 232 when calculating, except for*

*ml, the message length, which is a 64-bit quantity, and*

*hh, the message digest, which is a 160-bit quantity.*

*Note 2: All constants in this pseudo code are in* [*big endian*](https://en.wikipedia.org/wiki/Endianness)*.*

*Within each word, the most significant byte is stored in the leftmost byte position*

*Initialize variables:*

h0 = 0x67452301

h1 = 0xEFCDAB89

h2 = 0x98BADCFE

h3 = 0x10325476

h4 = 0xC3D2E1F0

ml = message length in bits (always a multiple of the number of bits in a character).

*Pre-processing:*

append the bit '1' to the message e.g. by adding 0x80 if message length is a multiple of 8 bits.

append 0 ≤ k < 512 bits '0', such that the resulting message length in *bits*

is congruent to −64 ≡ 448 (mod 512)

append ml, the original message length in bits, as a 64-bit big-endian integer.

Thus, the total length is a multiple of 512 bits.

*Process the message in successive 512-bit chunks:*

break message into 512-bit chunks

**for** each chunk

break chunk into sixteen 32-bit big-endian words w[i], 0 ≤ i ≤ 15

*Message schedule: extend the sixteen 32-bit words into eighty 32-bit words:*

**for** i **from** 16 to 79

*Note 3: SHA-0 differs by not having this leftrotate.*

w[i] = (w[i-3] **xor** w[i-8] **xor** w[i-14] **xor** w[i-16]) leftrotate 1

*Initialize hash value for this chunk:*

a = h0

b = h1

c = h2

d = h3

e = h4

*Main loop:*

**for** i **from** 0 **to** 79

**if** 0 ≤ i ≤ 19 **then**

f = (b **and** c) **or** ((**not** b) **and** d)

k = 0x5A827999

**else if** 20 ≤ i ≤ 39

f = b **xor** c **xor** d

k = 0x6ED9EBA1

**else if** 40 ≤ i ≤ 59

f = (b **and** c) **or** (b **and** d) **or** (c **and** d)

k = 0x8F1BBCDC

**else if** 60 ≤ i ≤ 79

f = b **xor** c **xor** d

k = 0xCA62C1D6

temp = (a **leftrotate** 5) + f + e + k + w[i]

e = d

d = c

c = b **leftrotate** 30

b = a

a = temp

*Add this chunk's hash to result so far:*

h0 = h0 + a

h1 = h1 + b

h2 = h2 + c

h3 = h3 + d

h4 = h4 + e

*Produce the final hash value (big-endian) as a 160-bit number:*

hh = (h0 **leftshift** 128) **or** (h1 **leftshift** 96) **or** (h2 **leftshift** 64) **or** (h3 **leftshift** 32) **or** h4

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ID | Chapter no | Description | Object Status | Object type | Requierement Type | Planned for release | Implemented in release | Author | Reviewer |
| 1 | 1.1 | Application | Complete | Header | Non functional | R2 | R2 | Maroiu Alexandru | Iordan Robert |
| 2 | 1.1.1. | System has 2 buttons and 2 LED’s. Pressing a button will toggle an LED. | Complete | Information | Non functional | R2 | R2 | Maroiu Alexandru | Iordan Robert |
| 3 | 2.1 | LED | Complete | Header | Non functional | R2 | R2 | Maroiu Alexandru | Iordan Robert |
| 4 | 2.1.1 | Designed PCB has 2 LED’s, left and right | Complete | Information | Non functional | R2 | R2 | Maroiu Alexandru | Iordan Robert |
| 5 | 2.1.2 | Left LED will be toggled by right button | Complete | Requirement | Functional | R2 | R2 | Maroiu Alexandru | Iordan Robert |
| 6 | 2.1.3 | Right LED will be toggled by left button | Complete | Requirement | Functional | R2 | R2 | Maroiu Alexandru | Iordan Robert |
| 7 | 2.1.4 | Left LED will be initialized with OFF state | Complete | Requirement | Functional | R2 | R2 | Maroiu Alexandru | Iordan Robert |
| 8 | 2.1.5 | Right LED will be initialized with OFF state | Complete | Requirement | Functional | R2 | R2 | Maroiu Alexandru | Iordan Robert |
| 9 | 3.1 | BUTTONS | Complete | Header | Non functional | R2 | R2 | Maroiu Alexandru | Iordan Robert |
| 10 | 3.1.1 | Designed system has 2 buttons, left and right | Complete | Information | Non functional | R2 | R2 | Maroiu Alexandru | Iordan Robert |
| 11 | 3.1.2 | Left Button shall toggle right LED | Complete | Requirement | Functional | R2 | R2 | Iordan Robert | Maroiu Alexandru |
| 12 | 3.1.3 | Right Button shall toggle left LED | Complete | Requirement | Functional | R2 | R2 | Iordan Robert | Maroiu Alexandru |
| 13 | 3.1.4 | Each button shall have 2 states, PRESSED or NOT PRESSED | Complete | Information | Non functional | R2 | R2 | Iordan Robert | Maroiu Alexandru |
| 14 | 3.1.5 | At startup, the buttons shall be initialized with a UNPRESSED state | Complete | Requirement | Non functional | R2 | R2 | Iordan Robert | Maroiu Alexandru |
| 15 | 3.1.6 | The debounce of the buttons stable state shall be 40ms. | Complete | Requirement | Functional | R2 | R2 | Iordan Robert | Maroiu Alexandru |
| 16 | 3.1.7 | The button module shall return always the last stable state | Complete | Requirement | Functional | R2 | R2 | Iordan Robert | Maroiu Alexandru |
| 17 | 4.1 | SECURITY | Complete | Header | Non functional | R2 | R2 | Iordan Robert | Maroiu Alexandru |
| 18 | 4.1.1 | The system will protect the button driver by verifying the hash calculated over the corresponding memory block of the driver | Complete | Information | Non functional | R2 | R2 | Iordan Robert | Maroiu Alexandru |
| 19 | 4.1.2 | The memory block start from 0x2000 and will be until 0x2800 address | Complete | Requirement | Non functional | R2 | R2 | Iordan Robert | Maroiu Alexandru |
| 20 | 4.1.3 | The hash algorith is SHA-1 and it will calculate a 160 bit hash | Complete | Requirement | Non functional | R2 | R2 | Iordan Robert | Maroiu Alexandru |
| 21 | 4.1.4 | The hash will be recalculated at every 10 ms | Complete | Requirement | Non functional | R3 | R3 | Iordan Robert | Maroiu Alexandru |
| 22 | 4.1.5 | În case of driver corruption, LED’s will be turned OFF | Complete | Requirement | Functional | R4 | R4 | Iordan Robert | Maroiu Alexandru |

|  |  |
| --- | --- |
| **Software architecture** | |
|  | |
| **BtnDrv Module** | |
| Description | Module that debounce and analyze the state of buttons to PRESSED or NOT\_PRESSED. It can be configured to have up to 8 buttons. |
| Interfaces with other components |  |
|  |  |
| Functions | void BtnDrvInit() --> this will set all buttons to the init state. |
|  | void BthDrvMain() --> the main function of the driver |
|  | BtnStateDrv BtnDrvGetButtonState(BtnDrvID) --> return the state of the button channel |
|  |  |
| TypeDef | teBtnDrvID (enum) --> ButtonLeft, ButtonRight, ButtonMaxNo |
|  | teBtnDrvState (enum) --> PRESSED, NOT\_PRESSED, INIT\_STATE |
| **LedDrv Module** | |
| Description | The LED driver can control up to 8 LED’s based on configuration. The LED’s can be turned on, off or toggle |
| Interfaces with other components |  |
|  |  |
| Functions | void LedDrvInit() -> will set the initial states of the LED driver |
|  | void LedDrvToggleLEDState(LedDrvID) -> will toggle the led based on parameter |
|  | void LedDrvSetLedState(LedDrvID, LedDrvState) -> will set a certain state of the LED based on parameters |
|  |  |
| TypeDef | LedDrvID(enum) --> ID of the LED’s (Left, Right, Max) |
|  | LedDrvState(enum) --> state of the LED’s: ON/OFF |
| **CySecDrv Module** | |
| Description | The software component shall calculate a hash of the driver module to verify the integrity of the driver |
| Interfaces with other components |  |
|  |  |
| Functions | Void CySecDrvInit() → initialization function of the CySecDrv component |
|  | void CySecDrvMain() → the main function that will calculate the hash |
|  | boolean CySecDrvIsHashValid() → fuction that will return if the calculated hash is correct or not |
|  |  |
| TypeDef | CySecDrvState (enum) -> security state of the system: SECURED/NOT\_SECURED |
|  |  |
| **Application Module** | |
| Description | Module that read the buttons state from the module and will toggle the LED’s state. It will toggle the LED’s if the hash is valid. |
| Interfaces with other components |  |
|  |  |
| Functions | void Application\_Init() → initialize the Application variables with init state |
|  | void Application\_Main() → the main function of the Application module, called by OS |
|  |  |
|  |  |
| TypeDef |  |
|  |  |