How to Implement SHA-1/HMAC Authentication for bq26100

Michael Vega *PMP Portable Power*

ABSTRACT

The bq26100 is a stand alone device that is used to authenticate peripherals when challenged by a system containing a microprocessor. An SHA-1 based HMAC is used to authenticate the bq26100. This document along with the Federal Information Processing Standards (FIPS) Publication 180-2, provides guidance to a firmware developer that must develop a driver to authenticate a peripheral containing a bq26100 with an expected 128-bit secret key.

Contents

1 SHA-1 Description 1
2 HMAC Description 2
3 Logic Symbols 2
4 Writing the SHA-1 Algorithm 2

List of Tables

1 Logic Symbols 2

1 SHA-1 Description

The SHA-1 is known as a one-way hash function, meaning there is no known mathematical method of computing the input given only the output. The specification of the SHA-1, as defined by Federal Information Processing Standards (FIPS) Publication 180-2, states that the input consists of 512 bit blocks with a total input length less than 264 bits. Inputs which do not conform to integer multiples of 512 bit blocks are padded before any block is input to the hash function. The SHA-1 algorithm outputs 160 bits, referred to as the digest. The full SHA-1 specification and algorithm are found at http://csrc.nist.gov/publications/fips under FIPS 180. (As of April 23, 2004 the latest revision is FIPS 180-2).

The bq26100 generates an SHA-1 input block of 288 bits (total input = 160 bit message +128 bit key). To complete the 512 bit block size requirement of the SHA-1, the bq26100 pads the key and message with a 1, followed by 159 0's, followed by the 64 bit value for 288 (000…00100100000), which conforms to the pad requirements specified by FIPS 180-2:

159 bits	64 bits
1 000 . . . 000	000 . . . 0100100000

2 HMAC Description

The SHA-1 engine is used to calculate a modified HMAC value. Using a public message and a secret key, the HMAC output is considered to be a secure *fingerprint* that authenticates the device used to generate the HMAC. To compute the HMAC let:

H designate the SHA-1 hash function

M designate the message transmitted to the bq26100,

K_D designate the unique 128 bit device key of the bq26100.

HMAC(M) is defined as:

$H[K_D \,||\, H(K_D \,||\, M)]$

where: || symbolizes an append operation. The message, *M*, is appended to the device key, K_D, and padded to become the input to the SHA-1 hash. The output of this first calculation is then appended to the device key, K_D, padded again, and cycled through the SHA-1 hash a second time.

The output is the HMAC digest value.

3 Logic Symbols

These are logic symbols that are used throughout this document to define functions.

Table 1. Logic Symbols

SYMBOL	DESCRIPTION	EXAMPLES (HEX FORMAT)
∧	Bitwise AND	ABCD1234 ∧ 567890EF = 02481024
∨	Bitwise OR	ABCD1234 ∨ 567890EF = FFFD92FF
¬	Bitwise complement	¬ABCD1234 = 5432EDCB
⊕	Bitwise XOR	ABCD1234 ⊕ 567890EF = FDB582DB
+	Addition Modulo 2^{32}	ABCD1234 + 567890EF = 0245A323
x << n	Shift x, n bits to the left	80008001 << 2 = 00020004
x >> n	Shift x, n bits to the right	80008001 >> 2 = 20002000

4 Writing the SHA-1 Algorithm

The information contained in this section is based on the SHA-1 specification in FIPS 180-2, but includes specific data based on the use of bq26100. The SHA-1/HMAC needed to authenticate the bq26100 requires running the SHA-1 algorithm twice. The whole process is described in this document. As an example, the secret key is in hexadecimal format

00000000 00000000 00000000 00000000

(which is not recommended in the actual application) and the message is “C82CA3CA 10DEC726 8E070A7C F0D1FE82 20AAD3B8”.

4.1 First Run of the SHA-1

4.1.1 Preprocessing I

The first step is to append the key with the message resulting as:

“00000000 00000000 00000000 00000000 C82CA3CA 10DEC726 8E070A7C F0D1FE82 20AAD3B8”

Then as described in the SHA-1 Description section of this document the key || message is padded with

“ 80000000 00000000 00000000 00000000 00000000 00000000 00000120”.

The bq26100 uses only a single 512-bit block message. To keep consistency with the FIPS 180-2 examples our padded key and message are broken down into the first 16 values of the message schedule as:

W_0 = 00000000
W_1 = 00000000
W_2 = 00000000
W_3 = 00000000
W_4 = C82CA3CA
W_5 = 10DEC726
W_6 = 8E070A7C
W_7 = F0D1FE82
W_8 = 20AAD3B8
W_9 = 80000000
W_{10} = 00000000
W_{11} = 00000000
W_{12} = 00000000
W_{13} = 00000000
W_{14} = 00000000
W_{15} = 00000120

From this section, what must be remembered by the firmware developer is that when authenticating with the bq26100 on the first run of the SHA-1, W_0 through W_3 correspond to the secret key, W_4 through W_8 correspond to the random message, and W_9 through W_{15} are always the same as with this example.

4.1.2 SHA-1 Computation I

The remaining 64 values of the message schedule are obtained with the function:

$$W_t = ROTL^1 (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) \text{ for } 16 \le t \le 79$$

In which $ROTL^n (x) = (x << n) \vee (x >> w - n)$.

Initialize the initial hash values and variables a, b, c, d, and e in hexadecimal format as follows:

H_0 = a = 67452301
H_1 = b = EFCDAB89
H_2 = c = 98BADCFE
H_3 = d = 10325476
H_4 = e = C3D2E1F0

Keep updating the variables a, b, c, d, and e with a Repeat For loop in which:

$$f_t(b,c,d) = \begin{cases} (b \wedge c) \oplus (\neg b \wedge d) & 0 \le t \le 19 \\ b \oplus c \oplus d & 20 \le t \le 39 \\ (b \wedge c) \oplus (b \wedge d) \oplus (c \wedge d) & 40 \le t \le 59 \\ b \oplus c \oplus d & 60 \le t \le 79 \end{cases}$$

$$K_t = \begin{cases} \text{5A827999} & 0 \le t \le 19 \\ \text{6ED9EBA1} & 20 \le t \le 39 \\ \text{8F1BBCDC} & 40 \le t \le 59 \\ \text{CA62C1D6} & 60 \le t \le 79 \end{cases}$$

For t = 0 to 79:

{

$Temp = ROTL^5(a) + f_t(b,c,d) + e + K_t + W_t$

$e = d$

$d = c$

$c = ROTL^{30}(b)$

$b = a$

$a = Temp$

}

Determine the hash value for the first run of SHA-1 by computing:

$H_0 = a + H_0$

$H_1 = b + H_1$

$H_2 = c + H_2$

$H_3 = d + H_3$

$H_4 = e + H_4$

The first 160-bit message digest is:

$H(K_D \| M) = H_0 \| H_1 \| H_2 \| H_3 \| H_4$

4.2 Second Run of the SHA-1

4.2.1 Preprocessing II

Having the $H(K_D \| M)$ portion of the bq26100 HMAC, on the second run through the SHA-1 computation, the $H(K_D \| M)$ portion replaces the M from the first run. Append the key with the $H(K_D \| M)$ result so that it is:

“ 00000000 00000000 00000000 00000000 EBF44E83 D792151C 8BE508BB 6D517C69 B331C0CE”

The padding for the $K_D \| H(K_D \| M)$ remains as:

“ 80000000 00000000 00000000 00000000 00000000 00000000 00000120”.

The first 16 values of the message schedule for the second run are:

W_0 = 00000000
W_1 = 00000000
W_2 = 00000000
W_3 = 00000000
W_4 = EBF44E83
W_5 = D792151C
W_6 = 8BE508BB
W_7 = 6D517C69
W_8 = B331C0CE
W_9 = 80000000
W_{10} = 00000000
W_{11} = 00000000
W_{12} = 00000000
W_{13} = 00000000
W_{14} = 00000000
W_{15} = 00000120

Notice how W_0 through W_3 remained the same because the key did not change.

4.2.2 SHA-1 Computation II

As with the first run of the SHA-1, the remaining 64 values of the message schedule are obtained with the function:

$W_t = ROTL^1(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16})$ for $16 \le t \le 79$

The initial hash values and variables a, b, c, d, and e are always initialized the same every time making a SHA-1 computation.

H_0 = a = 67452301
H_1 = b = EFCDAB89
H_2 = c = 98BADCFE
H_3 = d = 10325476
H_4 = e = C3D2E1F0

Keep updating the variables a, b, c, d, and e with the Repeat For loop in which:

$$f_t(b,c,d) = \begin{cases} (b \wedge c) \oplus (\neg b \wedge d) & 0 \le t \le 19 \\ b \oplus c \oplus d & 20 \le t \le 39 \\ (b \wedge c) \oplus (b \wedge d) \oplus (c \wedge d) & 40 \le t \le 59 \\ b \oplus c \oplus d & 60 \le t \le 79 \end{cases}$$

$$K_t = \begin{cases} \text{5A827999} & 0 \le t \le 19 \\ \text{6ED9EBA1} & 20 \le t \le 39 \\ \text{8F1BBCDC} & 40 \le t \le 59 \\ \text{CA62C1D6} & 60 \le t \le 79 \end{cases}$$

For t = 0 to 79:

{
Temp = $ROTL^5(a) + f_t(b,c,d) + e + K_t + W_t$
e = d
d = c
c = $ROTL^{30}(b)$
b = a
a = Temp
}

Determine the hash value for the second run of SHA-1 by computing:

$H_0 = a + H_0$
$H_1 = b + H_1$
$H_2 = c + H_2$
$H_3 = d + H_3$
$H_4 = e + H_4$

The final 160-bit message digest is:

$H[K_D \| H(K_D \| M)] = H_0 \| H_1 \| H_2 \| H_3 \| H_4$

Based on this example the response given by a bq26100 that contains:

128-bit key = 00000000000000000000000000000000, is challenged with

160-bit message = C82CA3CA10DEC7268E070A7CF0D1FE8220AAD3B8, is

FB8A342458E0B136988CB5203BB23F94DFD4440E.

This document has covered how to use the FIPS 180-2 publication for implementing the SHA-1/HMAC for the bq26100. Firmware developers may also use the example given on the FIPS 180-2 to debug their code. It is important to focus on making the first run of the SHA-1 work in the code given. The only difference with the second run is the values that are initialized into W_4 through W_8.