Document Version: 1.0.3 (January 21, 2003)

Notes on HAS-160

HAS-160 is a hash function designated for use with Korea's digital signature standard (which is related to DSA but has some differences). It is quite similar to SHA-1 but has some changes which supposedly increase the strength of the algorithm. Given the very wide adoption of SHA-1 (and now SHA-256 and SHA-512), it seems likely that HAS-160 will never get as much attention either in analysis or deployment than it may deserve. There were Internet Drafts which specified it's use as part of TLS, but these have long since expired.

HAS-160 is, however, significantly faster than SHA-1, and (at least at first glance) seems to be about as secure. The only obvious possible attack I see is the differential attack that broke SHA-0 (since HAS-160 does not do any rotations of the input during message expansion).

HAS-160 is somewhat similar to HAS-V, which was published in SAC 2000. HAS-160 can basically be seen as a product of a design transition that started with SHA-1 and ended up as HAS-V. It's used in some Korean products and is also available in Baltimore Technologies' KeyTools Crypto v5.0.1. There are several other hashes by the same designers which are similar, but not identical, including PMD-V, PMD-128, PMD-160, PMD-192, PMD-224, and PMD-256 (all of which are distinct algorithms). None of these have specifications in English, and I have been able to find very little information about them.

Since no description of HAS-160 is available in English, I'm attempting to describe it as best I can here. Keep in mind that I can't read Korean, so I'm basically using educated guesses, along with the fact that most of the diagrams and mathematics are fairly understandable. To make the most sense out of this, read FIPS 180-2 and the HAS-V paper first. The algorithm is largely described as the difference between it and SHA-1, so you must be reasonably familiar with that algorithm to understand this.

The full specification (in Korean) is available {here}.

I have implemented HAS-160 in C++ as part of the {Botan} crypto library. It is available in versions 1.1.1 and later.

If you have comments or questions, email lloyd (at) randombit (dot) net

---

Notation

In this document, 'step' refers to a single application of a sub-function, whereas 'round' refers to a block of them. For example, SHA-1 (and also HAS-160) uses 4 blocks of 20 steps each, for a total of 80 steps.

Description

HAS-160 uses a little endian ordering (unlike SHA-1), and has the same initial hash values (0x67452301, 0xEFCDAB89, etc) as in SHA-1.

Step Function

The main differences in the step function between SHA-1 and HAS-160 are as follows:

  HAS-160 has different round constants. Instead of
    0x5A827999, 0x6ED9EBA1, 0x8F1BBCDC, 0xCA62C1D6 (SHA-1)
  HAS-160 uses:
    0x00000000, 0x5A827999, 0x6ED9EBA1, 0x8F1BBCDC
Notice that the 2nd HAS-160 constant is the first SHA-1 constant, etc.

Boolean Functions: HAS-160, like SHA-1, has 4 3-term boolean functions. These are:
  * f0(x,y,z) == (x AND y) OR ((NOT x) AND z) [1], [2]
  * f1(x,y,z) == x XOR y XOR z [1]
  * f2(x,y,z) == y XOR (x OR (NOT Z)) [3]
  * f3(x,y,z) == x XOR y XOR z [1]
[1] Same as SHA-1
[2] This is equivalent to (z XOR (x AND (y XOR z)))
[3] SHA-1 uses (x AND y) OR (x AND z) OR (y AND z) here

Rotations

SHA-1 uses fixed rotations on A and B by 5 and 30 bits (respectively). HAS-160 uses two sets of rotation constants. One, the rotation of B, changes for each round (ie every 20 steps). The four rotation constants for B are 10, 17, 25, and 30.

Likewise, the fixed rotation of A by 5 bits in SHA-1 is replaced by a variable shift (one of 20 possible ones). These are used sequentially in the block, restarting the sequence at the start of the next block. The constants are: 5, 11, 7, 15, 6, 13, 8, 14, 7, 12, 9, 11, 8, 15, 6, 12, 9, 14, 5, 13

As a concrete example, in the 2nd step, A is rotated by 11 bits, and B is rotated by 10 bits. In the very last (80th) step, A is rotated by 13 bits and B is rotated by 30 bits.

Message Expansion

SHA-1 expands it's 16 word input into 80 words using a recursive definition. Like HAS-V, HAS-160 only derives a few extra words. These are denoted by X[16...19], and are changed at the start of each new round. They are produced by XORing together 4 of the original input words. The groupings are:
  Round 1: (0, 1, 2, 3), (4, 5, 6, 7), (8, 9, 10, 11), (12, 13, 14, 15)
  Round 2: (3, 6, 9, 12), (15, 2, 5, 8), (11, 14, 1, 4), (7, 10, 13, 0)
  Round 3: (12, 5, 14, 7), (0, 9, 2, 11), (4, 13, 6, 15), (8, 1, 10, 3)
  Round 4: (7, 2, 13, 8), (3, 14, 9, 4), (15, 10, 5, 0), (11, 6, 1, 12)
For example, in the second round, X[16...19] are derived by:
    X[16] = X[3] XOR X[6] XOR X[9] XOR X[12]
    X[17] = X[15] XOR X[2] XOR X[5] XOR X[8]
    X[18] = X[11] XOR X[14] XOR X[1] XOR X[4]
    X[19] = X[7] XOR X[10] XOR X[13] XOR X[0]
It is interesting to note that this is the same ordering used for processing the message, but with 16, 17, 18, and 19 removed from the sequences (see below).

Message Ordering

Within each round, there is a separate message ordering where each of the 20 values in X[] are processed. The message orders are:
  Round 1: 18, 0, 1, 2, 3, 19, 4, 5, 6, 7, 16, 8, 9, 10, 11, 17, 12, 13, 14, 15
  Round 2: 18, 3, 6, 9, 12, 19, 15, 2, 5, 8, 16, 11, 14, 1, 4, 17, 7, 10, 13, 0
  Round 3: 18, 12, 5, 14, 7, 19, 0, 9, 2, 11, 16, 4, 13, 6, 15, 17, 8, 1, 10, 3
  Round 4: 18, 7, 2, 13, 8, 19, 3, 14, 9, 4, 16, 15, 10, 5, 0, 17, 11, 6, 1, 12
There is a nice table on page 11 of the PDF (marked as page 7) which shows this ordering.

HAS-160 Test Vectors

These test vectors were taken from TTAS.KO-12.0011/R1

The specification also (thankfully) includes some intermediate value examples for a couple of different messages. These are quite understandable if you're familiar with, for example, FIPS 180-2 (since they're simply hex dumps, the fact that the rest of the paper is in a foreign language doesn't matter). These are not included here because my PDF reading software seems unable to handle copying them out, meaning they would have to be typed out by hand (as these test vectors were).
HAS-160("") =
  30 79 64 EF 34 15 1D 37 C8 04 7A DE C7 AB 50 F4 FF 89 76 2D

HAS-160("a") =
  48 72 BC BC 4C D0 F0 A9 DC 7C 2F 70 45 E5 B4 3B 6C 83 0D B8

HAS-160("abc") =
  97 5E 81 04 88 CF 2A 3D 49 83 84 78 12 4A FC E4 B1 C7 88 04

HAS-160("message digest") =
  23 38 DB C8 63 8D 31 22 5F 73 08 62 46 BA 52 9F 96 71 0B C6

HAS-160("abcdefghijklmnopqrstuvwxyz") =
  59 61 85 C9 AB 67 03 D0 D0 DB B9 87 02 BC 0F 57 29 CD 1D 3C

HAS-160("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789") =
  CB 5D 7E FB CA 2F 02 E0 FB 71 67 CA BB 12 3A F5 79 57 64 E5

```
HAS-160(eight of "1234567890") =
  07 F0 5C 8C 07 73 C5 5C A3 A5 A6 95 CE 6A CA 4C 43 89 11 B5

HAS-160(a million of "a") =
  D6 AD 6F 06 08 B8 78 DA 9B 87 99 9C 25 25 CC 84 F4 C9 F1 8D
HMAC(HAS-160) Test Vectors

These were found in an expired Internet draft (draft-ietf-tls-seedhas-00.txt).
The format is that used in all of the HMAC RFCs.
   test_case =     1
   key =           0x0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b
   key_len =       20
   data =          "Hi There"
   data_len =      8
   digest =        0xf5b44115a53f716b6f488de1098ee7c251418623

   test_case =     2
   key =           "Jefe"
   key_len =       4
   data =          "what do ya want for nothing?"
   data_len =      28
   digest =        0xa74547c1ef0aa147c7428ab7e71664549be2a412

   test_case =     3
   key =           0xaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
   key_len =       20
   data =          0xdd repeated 50 times
   data_len =      50
   digest =        0xe4c91bc71782fa44a56be1a34aae167e8ffc9734

   test_case =     4
   key =           0x0102030405060708090a0b0c0d0e0f10111213141516171819
   key_len =       25
   data =          0xcd repeated 50 times
   data_len =      50
   digest =        0x14d1055da875222053bf1180bbef8892eba3ac30

# WARNING: This seems to be incorrect (same output as test case #6 [typo]).
#          My implementation produces 124131A293F1FDF3D6B11E2B7F7A1F5B12E42D58
   test_case =     5
   key =           0x0c0c0c0c0c0c0c0c0c0c0c0c0c0c0c0c0c0c0c0c
   key_len =       20
   data =          "Test With Truncation"
   data_len =      20
   digest =        0x63750d67af40e3fde33526545d300972a1527053

   test_case =     6
   key =           0xaa repeated 80 times
   key_len =       80
   data =          "Test Using Larger Than Block-Size Key - Hash Key First"
   data_len =      54
   digest =        0x63750d67af40e3fde33526545d300972a1527053

   test_case =     7
   key =           0xaa repeated 80 times
   key_len =       80
   data =          "Test Using Larger Than Block-Size Key and Larger
                   Than One Block-Size Data"
   data_len =      73
   digest =        0x1bdb821e399e208352c64f0655f6601e2a8a087c

------------------------------------
Revision History
  * October 1, 2002 (1.0.0)
    * First version
  * October 2, 2002 (1.0.1)
    * Clarified some of the text
    * Fixed a test vector typo
    * Added one more test vector
    * Added note that HMAC test #5 seems to be incorrect
    * Corrected description of message expansion (was incomplete)
  * January 11, 2003 (1.0.2)
```

```
    * Reformatted to HTML (no text changes)
  * January 22, 2003 (1.0.3)
    * Fixed some minor formatting errors
    * Minor text changes

-----------------------------------
```