Description of Known Answer Test (KAT) and Monte Carlo Test (MCT) for SHA-3 Candidate Algorithm Submissions

Revision 3: February 20, 2008

1. Overview

Each submission package is required to include Known Answer Test (KAT) and Monte Carlo Test (MCT) values, which can be used to determine the correctness of an implementation of the candidate algorithm. Values shall be included (at a minimum) for each of the four minimum required hash sizes: 224, 256, 384, and 512-bits.

These KAT and MCT tests are based on tests specified in The Secure Hash Algorithm Validation System (SHAVS) [SHAVS], which describes tests for the SHA-2 family of hash functions. Each of the tests for which values are required in the submission packages is described below. In addition, example files are included which specify the exact syntax and format that submitters are required to use when submitting their KAT and MCT values.

2. Known Answer Tests (KAT)

Known Answer Test values must be provided with submissions, which demonstrate operation of the SHA-3 candidate algorithm with varying length inputs, for each of the minimum required hash length values (224, 256, 384, and 512-bits). There are three types of KATs that are required for all submissions: 1) Short Message Test, 2) Long Message Test, and 3) Extremely Long Message Test.

The file formats for the first two tests are identical. There is some header information that is used to identify the algorithm for which the samples pertain followed by a sequence of sample values. Each sequence consists of 3 lines. The first line provides the length, in bits, of the message. The second line provides the message in hexadecimal digits; it is the most significant, or leftmost, number of bits as specified in *length* of the provided string. And the third line will contain the message digest of the provided message. Two sample sequences, using SHA-256, follow:

```
Len = 2
Msg = c0
MD = 1e1cea10a23697dc97b423c259842ac12ee679d6b43f088f3c32b26dbbfb0d79

Len = 3
Msg = c0
MD = fa0e40cc693c20d55b131b825a32f961d6d0681811a95886d6704e9c376a9abd
```

The first sample message is 2 bits in length and is a 1-bit followed by another 1-bit. The second sample message is 3 bits in length and is a 1-bit, followed by a 1-bit, followed by a 0-bit.

The file format for the third test is slightly different in that the message is too long to enumerate fully in the provided file. The details of how the file format differs for the Extremely Long Message Test are provided in the test description below.

2.1. Short Message Test

Each SHA-3 candidate algorithm must be able to generate message digests for messages of arbitrary length. To test this, the candidate algorithm must supply message digest for random messages whose length ranges from 0 to 2047-bits. The source file with the messages to be hashed will be provided on the Hash Competition/Submission Requirements/Test Vectors page (http://www.nist.gov/hash-competition) and will be called *ShortMsgKAT.txt*. The source file should be processed four times, once for each required hash length, replacing the question marks on the "MD = ??" line with the appropriate message digest. The resulting file, with the message digests for a particular hash length, shall be provided in a file called *ShortMsgKAT_{Length}.txt* (for example, the file with hash length 256 will be provided in a file called *ShortMsgKAT_256.txt*). Therefore, there should be four files provided for this test.

2.2. Long Message Test

To test an algorithm's ability to process long messages, and therefore properly handle chaining variables, the candidate algorithm must supply message digests for a sampling of long messages. The messages will be supplied in a file called *LongMsgKAT_{Length}.txt* for each of the required hash length. For example, the file to test for hash length 256 will be provided in *LongMsgKAT_256.txt*.

2.3. Extremely Long Message Test

To test an algorithm's ability to process messages longer that 2^{32}-bits in length, the candidate must supply a message digest for one extremely long sequence. The sequence is a 64-byte character string repeated 16,777,216 times. The message will be supplied in a file called *ExtremelyLongMsgKAT_{Length}.txt* for each of the required hash length. For example, the file to test for hash length 256 will be provided in *ExtremelyLongMsgKAT_256.txt*.

The file will have the same header portion as the other two KAT test, but the data portion will be different. There will only be one sample for each hash length.

```
Repeat = 16777216
Text = abcdefghbcdefghicdefghijdefghijkefghijklfghijklmghijklmnhijklmno
MD = 50e72a0e26442fe2552dc3938ac58658228c0cbfb1d2ca872ae435266fcd055e
```

The "Repeat" line tells how many times to repeat the "Text" line. The "Text" line provides the 64-byte ASCII character string that will be repeated. Finally, the "MD" line provides the message digest of the repeated string.

3. Monte Carlo Test

The Monte Carlo Test provides a way to stress the internal components of a candidate algorithm. A seed message will be provided. This seed is used by a pseudorandom function together with the candidate algorithm to generate 100,000 message digests. 100 of these 100,000 message digests, i.e. once every 1,000 hashes, are recorded as checkpoints to the operation of the candidate algorithm. The procedure used to generate the 100 checkpoint message digests is as follows:

1) 100,000 pseudorandom messages are generated by using previous message digests as the input to the candidate hash algorithm (see Figure 1 for a description of how the pseudorandom

messages are generated); and,

2) After every 1,000 hashes a sample is taken and is provided as a checkpoint.

```
Input:
      Seed – A random seed of 1024 bits
      hashbitlen – the length, in bits, of the message digests being tested
{
      Msg = Seed;
      for (j=0; j<100; j++) {
            for (i=0; i<1000; i++) {
                  Hash(hashbitlen, Msg, 1024, MD);
                  Temp = MSB1024-hashbitlen(Msg);
                  Msg = MD || temp;
            }
            Output: j, MD
      }
}

MSB – Most Significant Bits
|| - Concatenation

As an example, let hashbitlen be 256.  The next pseudorandom message is derived by
concatenating the most significant 1024-256=768 bits of the current message to the 256-
bit message digest of the current message.
```

Figure 0 Monte Carlo Test Pseudocode

The messages to be hashed will be supplied in a file called *MonteCarlo_{Length}.txt* for each of the required hash length. For example, the file to test for hash length 256 will be provided in *MonteCarlo_256.txt*.

4. File Formats

The following shows the format of the files for each of the tests.

4.1. Short Message and Long Message Tests

The file formats for the source file and the processed KAT file of both the **Short Message Test** and the **Long Message Test** are identical. The source file, in a format illustrated in Section 4.1.1, will be provided by NIST. The processed data, in the same format as illustrated in Section 4.1.2, shall be provided by the submitter in their submission package.

4.1.1. Source File

```
# ShortMsgKAT.txt
# Algorithm Name: (as stated on the submission cover sheet)
# Principal Submitter: (as stated on the submission cover sheet)

Len = 0
Msg = 00
MD = ??

Len = 1
```

```
Msg = 00
MD = ??

Len = 2
Msg = c0
MD = ??

Len = 3
Msg = c0
MD = ??

Len = 4
Msg = 80
MD = ??

Len = 5
Msg = 48
MD = ??

…

Len = 512
Msg =
e926ae8b0af6e53176dbffcc2a6b88c6bd765f939d3d178a9bde9ef3aa131c61e31c1e42cdfaf4b4dcde579a3
7e150efbef5555b4c1cb40439d835a724e2fae7
MD = ??
…
```

4.1.2. Processed KAT File from the Submitter

```
# ShortMsgKAT_256.txt
# Algorithm Name: SHA-256
# Principal Submitter: John Q Cryptographer

Len = 0
Msg = 00
MD = e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855

Len = 1
Msg = 00
MD = bd4f9e98beb68c6ead3243b1b4c7fed75fa4feaab1f84795cbd8a98676a2a375

Len = 2
Msg = c0
MD = 1e1cea10a23697dc97b423c259842ac12ee679d6b43f088f3c32b26dbbfb0d79

Len = 3
Msg = c0
MD = fa0e40cc693c20d55b131b825a32f961d6d0681811a95886d6704e9c376a9abd

Len = 4
Msg = 80
MD = c519acdb14daa2a091c85bb8578e95614d429b0c96296e675649768ae2a3c706

Len = 5
Msg = 48
```

```
MD = 0aafa19cea5ebf0a04a5add2d64c870ff8377164f403895a51f6b00f07af7b6f

…

Len = 512
Msg =
e926ae8b0af6e53176dbffcc2a6b88c6bd765f939d3d178a9bde9ef3aa131c61e31c1e42cdfaf4b4dcde579a3
7e150efbef5555b4c1cb40439d835a724e2fae7
MD = f4a1a02e8e35822107cc08b5b21b661c058e2825f3c7cb2a3b6d40f3ae89211b
…
```

4.2. Extremely Long Message Test

4.2.1. Source File

```
# ExtremelyLongMsgKAT.txt
# Algorithm Name: (as stated on the submission cover sheet)
# Principal Submitter: (as stated on the submission cover sheet)

Repeat = 16777216
Text = abcdefghbcdefghicdefghijdefghijkefghijklfghijklmghijklmnhijklmno
MD = ??
```

4.2.2. Processed KAT File from the Submitter

```
# ExtremelyLongMsgKAT_256.txt
# Algorithm Name: SHA-256
# Principal Submitter: John Q Cryptographer

Repeat = 16777216
Text = abcdefghbcdefghicdefghijdefghijkefghijklfghijklmghijklmnhijklmno
MD = 50e72a0e26442fe2552dc3938ac58658228c0cbfb1d2ca872ae435266fcd055e
```

4.3. Monte Carlo Test

4.3.1. Source File

```
# MonteCarlo.txt
# Algorithm Name: (as stated on the submission cover sheet)
# Principal Submitter: (as stated on the submission cover sheet)

Seed =
6cd4c0c5cb2ca2a0f1d1aecebac03b52e64ea03d1a1654372936545b92bbc5484a59db74bb60f9c40ceb1a5aa
35a6fafe80349e14c253a4e8b1d77612ddd81ace926ae8b0af6e53176dbffcc2a6b88c6bd765f939d3d178a9b
de9ef3aa131c61e31c1e42cdfaf4b4dcde579a37e150efbef5555b4c1cb40439d835a724e2fae7
```

4.3.2. Sample Values

```
# MonteCarlo_256.txt
# Algorithm Name: SHA-256
# Principal Submitter: John Q. Cryptographer

Seed =
6cd4c0c5cb2ca2a0f1d1aecebac03b52e64ea03d1a1654372936545b92bbc5484a59db74bb60f9c40ceb1a5aa
```

```
35a6fafe80349e14c253a4e8b1d77612ddd81ace926ae8b0af6e53176dbffcc2a6b88c6bd765f939d3d178a9b
de9ef3aa131c61e31c1e42cdfaf4b4dcde579a37e150efbef5555b4c1cb40439d835a724e2fae7

j = 0
MD = 4232dbc92aaf0e27a4d7f3d929430943541b2c68299809a1874cd8adee7002d8

j = 1
MD = cf499d3094136814943410215aee77e951ff1a41adfbba3a8899720b05e101d4

…

j = 99
MD = 45f72e1ab055e54f5c26aebe73bb7f4ec8f795c9d4f94e3f258ed130ab3d3347
```

5. References

[SHAVS] *The Secure Hash Algorithm Validation System (SHAVS)*, July 22, 2004, http://csrc.nist.gov/groups/STM/cavp/documents/shs/SHAVS.pdf .