The Secure Hash Algorithm Validation System (SHAVS)

Updated: May 21, 2014
Created: July 22, 2004

Lawrence E. Bassham III

Timothy A. Hall

National Institute of Standards and Technology

Information Technology Laboratory

Computer Security Division

TABLE OF CONTENTS

Update Log

5/21/14

- Changed “SHA-1” to “SHA=256” in first paragraph of Section 6.3.2: “Therefore, for testing SHA-1…” should be “Therefore, for testing SHA-256…”

7/23/12

- Updated References to FIPS 180-4 throughout document.
- Changed “Pseudorandomly Generated Messages Test” to “Pseudorandomly Generated Messages (Monte Carlo) Test” throughout document.
- Minor corrections and edits.

1 Introduction

This document, *The Secure Hash Algorithm Validation System (SHAVS)* specifies the procedures involved in validating implementations of the Secure Hash Algorithms in FIPS 180-4, *Secure Hash Standard* [1]. The SHAVS is designed to perform automated testing on Implementations Under Test (IUTs). This document provides the basic design and configuration of the SHAVS. It includes the specifications for the three categories of tests that make up the SHAVS, i.e., the Short Messages Test, Long Messages Test, and Pseudorandomly Generated Messages (Monte Carlo) Test. The requirements and administrative procedures specific to those seeking formal validation of an implementation of the Secure Hash Algorithm(s) are presented. The requirements described include the specific protocols for communication between the IUT and the SHAVS, the types of tests that the IUT must pass for formal validation, and general instructions for accessing and interfacing with the SHAVS.

2 Scope

This document specifies the tests required to validate IUTs for conformance to the Secure Hash Algorithm(s) as specified in [1]. When applied to IUTs that implement SHA, the SHAVS provides testing to determine the correctness of the algorithm implementation. In addition to determining conformance, the SHAVS is structured to detect implementation flaws including pointer problems, insufficient allocation of space, improper error handling, and incorrect behavior of the SHA implementation.

The SHAVS is composed of three types of validation tests, the Short Message Test, the Long Message Test, and the Pseudorandomly Generated Messages (Monte Carlo) test. Additionally, the first two tests have an option to test bit-oriented or byte-oriented implementations. Byte-oriented implementations only hash messages with lengths divisible by 8, or integral bytes worth of data. Bit-oriented implementations can handle any length message (up to the limitation of the particular algorithm). While the specification for SHA specifies that messages up to at least $2^{64} - 1$ bits are possible, these tests only test messages up to a limited size of approximately 100,000 bits. This is adequate for detecting algorithmic and implementation errors.

3 Conformance

The successful completion of the tests contained within the SHAVS is required to claim conformance to FIPS 180-4. Testing for the cryptographic module in which the various algorithms specified in FIPS 180-4 is implemented is defined in FIPS PUB 140-2, *Security Requirements for Cryptographic Modules* [2].

4 Definitions, Symbols, and Abbreviations

4.1 Definitions

DEFINITION	MEANING
CST laboratory	Cryptographic Security Testing laboratory that operates the SHAVS
Secure Hash Algorithm(s)	The algorithm(s) specified in FIPS 180-4, *Secure Hash Standard (SHS)*.

4.2 Symbols

SYMBOL	MEANING
L	The length of the message digest in bytes
Len	The length of a message in bits
l_{min}	Minimum message length for Selected Long Message Test
l_{max}	Maximum message length for Selected Long Message Test
m	Number of bits in the message block
MD	A Message Digest
Msg	A message
n	Number of bits in the message digest

4.3 Abbreviations

ABBREVIATION	MEANING
IUT	Implementation Under Test
SHA	Secure Hash Algorithm(s) specified in FIPS 180-4
SHAVS	Secure Hash Algorithm Validation System

5 Design Philosophy Of The Secure Hash Algorithm Validation System

The SHAVS is designed to test conformance to SHA rather than provide a measure of a product's security. The validation tests are designed to assist in the detection of accidental implementation errors, and are not designed to detect intentional attempts to misrepresent conformance. Thus, validation should not be interpreted as an evaluation or endorsement of overall product security.

The SHAVS has the following design philosophy:

1. The SHAVS is designed to allow the testing of an IUT at locations remote to the SHAVS. The SHAVS and the IUT communicate data via *REQUEST* and *RESPONSE* files.

2. The testing performed within the SHAVS utilizes statistical sampling (i.e., only a small number of the possible cases are tested); hence, the successful validation of a device does not imply 100% conformance with the standard.

6 SHAVS Tests

The SHAVS for the Secure Hash Algorithm(s) consists of three types of tests: The Short Messages Test, The Long Messages Test, and The Pseudorandomly Generated Messages (Monte Carlo) Test. The SHAVS provides conformance testing for the algorithm, as well as testing for apparent implementation errors. The IUT may be implemented in software, firmware, hardware, or any combination thereof.

An IUT may implement SHA in either of two modes. The first mode is the byte-oriented mode. In this mode the IUT only hashes messages that are an integral number of bytes in length; i.e., the length (in bits) of the message to be hashed is divisible by 8. The second mode is the bit-oriented mode. In this mode the IUT hashes messages of arbitrary length. Selecting the proper mode for an implementation will determine how the SHAVS tests will be performed. Both modes can be selected for the Short Messages Test and the Selected Long Messages Test. The Pseudorandomly Generated Messages (Monte Carlo) Test is always run in byte-oriented mode.

6.1 Configuration Information

To initiate the validation process of the SHAVS, a vendor submits an application to an accredited laboratory requesting the validation of their implementation of SHA. The vendor's implementation is referred to as the Implementation Under Test (IUT). The request for

validation includes background information describing the IUT along with information needed by the SHAVS to perform the specific tests. More specifically, the request for validation should include:

1. Vendor Name;
2. Product Name;
3. Product Version;
4. Implementation in software, firmware, or hardware;
5. Processor and Operating System with which the IUT was tested if the IUT is implemented in software or firmware;
6. Brief description of the IUT or the product/product family in which the IUT is implemented by the vendor (2-3 sentences); and
7. Whether the IUT handles bit-oriented messages or only byte-oriented messages.

6.2 The Short Messages Test

An implementation of SHA must be able to correctly generate message digests for messages of arbitrary length. The SHAVS tests this property by supplying the IUT with a number of short messages. The Short Messages Test has two versions, one for bit-oriented implementations and another for byte-oriented implementations.

6.2.1 The Short Messages Test for Bit-Oriented Implementations

This test generates a number of short messages equal to the number of bits in the hash block plus one. For example, SHA-256 defines a block length of m=512 bits. Therefore, for testing SHA-256, 513 unpredictable messages will be generated with lengths from 0 to 512 bits.

The SHAVS:

A. Generates $m + 1$ messages of length 0 to m.
B. Creates a *REQUEST* file (Filename: <Alg>ShortMsg.req) containing:
    1. The Product Information (vendor, product name, version); and
    2. The sequence of $m + 1$ messages to be hashed.

    Note: The CST laboratory sends the *REQUEST* file to the IUT.
C. Creates a *FAX* file (Filename: <Alg>ShortMsg.fax) containing:
    1. The Product Name; and
    2. The $m + 1$ datasets containing:
        a. The messages, and

b. The message digest of the message.

Note: The CST laboratory retains the *FAX* file.

The IUT:

A. Generates message digests using the messages supplied by the SHAVS in the *REQUEST* file.

B. Creates a *RESPONSE* file (Filename: <Alg>ShortMsg.rsp) containing:

1. The Product Name; and
2. The m + 1 datasets containing:
   a. The messages, and
   b. The message digest of the messages.

Note: The IUT sends the *RESPONSE* file to the SHAVS.

The SHAVS:

A. Compares the *RESPONSE* file with the *FAX* file. For each message, the SHAVS verifies that the IUT generated the correct message digest.

B. If all message digests generated by the IUT are correct, records PASS for this test; otherwise, records FAIL.

6.2.2 The Short Messages Test for Byte-Oriented Implementations

This test generates a number of short messages equal to the number of bytes in the hash block plus one. For example, SHA-256 defines a block length of m=512 bits resulting in a block size of 64 bytes. Therefore, for testing SHA-256, 65 unpredictable messages will be generated with lengths of 0, 8, 16, …, 512 bits.

The SHAVS:

A. Generates $m/8 + 1$ messages of length 0, 8, 16, …, m.

B. Creates a *REQUEST* file (Filename: <Alg>ShortMsg.req) containing:

1. The Product Name; and
2. The sequence of $m/8 + 1$ messages to be hashed.

Note: The CST laboratory sends the *REQUEST* file to the IUT.

C. Creates a *FAX* file (Filename: <Alg>ShortMsg.fax) containing:

1. The Product Name; and
2. The $m/8 + 1$ datasets containing:
   a. The message, and

b. The message digest of the message.

Note: The CST laboratory retains the *FAX* file.

The IUT:

A. Generates message digests using the messages supplied by the SHAVS in the *REQUEST* file.

B. Creates a *RESPONSE* file (Filename: <Alg>ShortMsg.rsp) containing:

1. The Product Name; and

2. The $m/8 + 1$ datasets containing:

a. The message, and

b. The message digest of the message.

Note: The IUT sends the *RESPONSE* file to the SHAVS.

The SHAVS:

A. Compares the *RESPONSE* file with the *FAX* file. For each message, the SHAVS verifies that the IUT generated the correct message digest.

B. If all message digests generated by the IUT are correct, records PASS for this test; otherwise, records FAIL.

6.3 The Selected Long Messages Test

An implementation of SHA must be able to correctly generate message digests for messages that span multiple message blocks. The SHAVS tests this property by supplying selected unpredictable long messages to the IUT. The IUT then generates message digests for each message. The Selected Long Messages Test has two versions, one for bit-oriented implementations and another for byte-oriented implementations.

6.3.1 The Selected Long Messages Test for Bit-Oriented Implementations

This test generates a number of long messages equal to the number of bits in the hash block, *m*. These message range in size from $m+99 \le len \le m*100$. For example, SHA-256 defines a block length of $m=512$ bits. Therefore, for testing SHA-256, 512 unpredictable long messages will be generated with lengths (in bits) of:

$512 + 99*i$, $1 \le i \le 512$.

The SHAVS:

A. Generates *m* messages of the length specified above.

B. Creates a *REQUEST* file (Filename: <Alg>LongMsg.req) containing:

1. The Product Name; and

2. The sequence of *m* messages to be hashed.

Note: The CST laboratory sends the *REQUEST* file to the IUT.

C. Creates a *FAX* file (Filename: <Alg>LongMsg.fax) containing:

1. The Product Name; and
2. The *m* datasets containing:
   a. The message, and
   b. The message digest of the message.

Note: The CST laboratory retains the *FAX* file at the SHAVS.

The IUT:

A. Generates message digests using the messages supplied by the SHAVS in the *REQUEST* file.

B. Creates a *RESPONSE* file (Filename: <Alg>LongMsg.rsp) containing:

1. The Product Name; and
2. The *m* datasets containing:
   a. The message, and
   b. The message digest of the message.

Note: The IUT sends the *RESPONSE* file to the SHAVS.

The SHAVS:

A. Compares the *RESPONSE* file with the *FAX* file. For each message, the SHAVS verifies that the IUT generated the correct message digest.

B. If all message digests generated by the IUT are correct, records PASS for this test; otherwise, records FAIL.

6.3.2 The Selected Long Messages Test for Byte-Oriented Implementations

This test generates a number of long messages equal to the number of bytes in the hash block, $m/8$. These message range in size from $m+99 \le len \le m*100$. For example, SHA-256 defines a block length of $m/8 = 64$ bytes. Therefore, for testing SHA-256, 64 unpredictable long messages will be generated with lengths (in bits) of:

$512 + 8*99*i$, $1 \le i \le 64$.

The SHAVS:

A. Generates $m/8$ messages of the length specified above.

B. Creates a *REQUEST* file (Filename: LongMsg.req) containing:

1. The Product Information (vendor, product name, version); and
2. The sequence of *m*/8 messages to be hashed.

Note: The CST laboratory sends the *REQUEST* file to the IUT.

C. Creates a *FAX* file (Filename: LongMsg.fax) containing:

1. The Product Information (vendor, product name, version); and
2. The *m*/8 datasets containing:
   a. The message, and
   b. The message digest of the message.

Note: The CST laboratory retains the *FAX* file.

The IUT:

A. Generates message digests using the message supplied by the SHAVS in the *REQUEST* file.

B. Creates a *RESPONSE* file (Filename: LongMsg.rsp) containing:

1. The Product Information (vendor, product name, version); and
2. The *m*/8 datasets containing:
   a. The message, and
   b. The message digest of the message.

Note: The IUT sends the *RESPONSE* file to the SHAVS.

The SHAVS:

A. Compares the *RESPONSE* file with the *FAX* file. For each message, the SHAVS verifies that the IUT generated the correct message digest.

B. If all message digests generated by the IUT are correct, records PASS for this test; otherwise, records FAIL.

6.4 The Pseudorandomly Generated Messages (Monte Carlo) Test

The SHAVS tests the correctness of message digests generated from pseudorandomly generated messages by supplying a seed, *Seed*, of length *n* bits. This seed is used by a pseudorandom function to generate 100,000 message digests. 100 of the 100,000 message digests, once every 1,000 hashes, are recorded as checkpoints to the operation of the generator. The IUT uses the same procedure to generate the same 100,000 message digests and 100 checkpoint values. The SHAVS compares each of the recorded 100 message digests with those generated by the IUT.

The procedure used to generate the 100 checkpoint messages digest is as follows:

1) 100,000 pseudorandom messages are generated by using previous message digests as the input to the hash algorithm; and,

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

```
INPUT: Seed  - A random seed n bits long
{
     for (j=0; j<100; j++) {
          MD0 = MD1 = MD2 = Seed;
          for (i=3; i<1003; i++) {
               Mi = MDi-3 || MDi-2 || MDi-1;
               MDi = SHA(Mi);
               }
          MDj = Seed = MD1002;
          OUTPUT: MDj
          }
     }
```

These checkpoints are denoted MD_j in Figure 1.

Figure 1: Code for Generating Pseudorandom Messages

The SHAVS:

A. Generates a seed, *Seed*, of length *n* bits.

B. Creates a *REQUEST* file (Filename: Monte.req) containing:

   1. The Product Information (vendor, product name, version); and
   2. The *Seed*.

   Note: The CST laboratory sends the *REQUEST* file to the IUT.

C. Creates a *FAX* file (Filename: Monte.fax) containing:

   1. The Product Information (vendor, product name, version);
   2. The *Seed*; and
   3. The 100 message digests, MD_j, from Figure 1.

   Note: The CST laboratory retains the *FAX* file at the SHAVS.

The IUT:

A. Generates the 100 message digest using the *Seed* supplied by the SHAVS in the *REQUEST* file.

B. Creates a *RESPONSE* file (Filename: Monte.rsp) containing:

   1. The Product Information (vendor, product name, version);
   2. The *Seed*; and
   3. The 100 message digest MD_j from Figure 1.

   Note: The IUT sends the *RESPONSE* file to the SHAVS.

The SHAVS:

A. Compares the *RESPONSE* file with the *FAX* file. The SHAVS verifies that the IUT generated the correct message digests.

B. If all message digests generated by the IUT are correct, records PASS for this test; otherwise, records FAIL.

Appendix A References

[1] *Secure Hash Standard (SHS)*, FIPS Publication 180-4, National Institute of Standards and Technology, March 2012.

[2] *Security Requirements for Cryptographic Modules*, FIPS Publication 140-2, National Institute of Standards and Technology, May 2001.