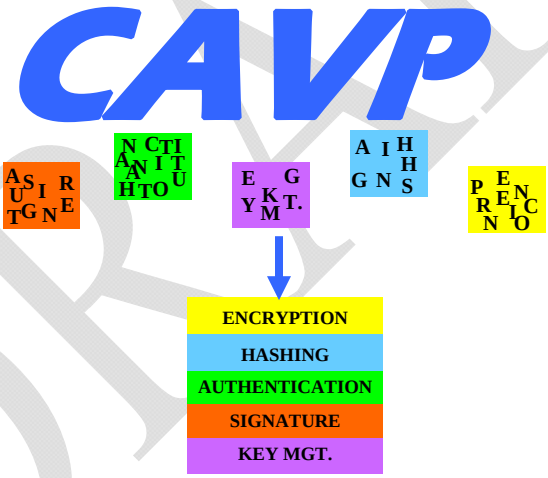
Frequently Asked Questions
For the
Cryptographic Algorithm Validation Program
Concerning the Validation of Cryptographic Algorithm Implementations

National Institute of Standards and Technology
Communications Security Establishment



Initial Release: May 16, 2005

Previous Update: November 20, 2007

Previous Update: January 2, 2008

Last Update: June 11, 2008

New FAQ's and Modified FAQ's (Issued within the last 45 days)

New FAQ's

Modified FAQ's

- **6-10-08 GEN.2 Added text to answer.**
- **6-10-08 GEN.3 Changing question to be "How should the algorithm implementation be named?" Also changing answer.**
- **6-10-08 GEN.9 The operating environment(s) listed for a validated cryptographic algorithm implementation tested under the CAVP indicates the operating environment(s) on which the implementation was tested.**
- **6-10-08 GEN.11 Added text to answer.**

1 Introduction....4
2 General Algorithm FAQs (Can be applied to all algorithms)....5
3 AES FAQ....14
4 DES FAQ....16
5 Triple-DES FAQ....17
6 DSA FAQ....18
7 SHA FAQ....21
8 RNG FAQ....22
9 RSA FAQ....23
10 HMAC FAQ....27
11 CCM FAQ....29
12 ECDSA FAQ....30
13 CMAC FAQ....32

1 Introduction

Below is a compilation of questions received from the Cryptographic Module Testing (CMT) laboratories relating to the validation of cryptographic algorithm implementations. Some of the questions are related to the Cryptographic Algorithm Validation Program (CAVP) while others are related to the Cryptographic Algorithm Validation System (CAVS) tool and how it is used to validate these implementations.

This is intended for use by the CMT laboratories when validating cryptographic algorithms submitted by vendors. Vendors may find the information useful when submitting their information to the CMT laboratories for cryptographic algorithm implementation validation. This compilation of topics covers issues such as what information is required when validating an implementation, individual cryptographic algorithm guidance, how to use the CAVS tool, etc.

Currently the CAVP provides validation testing for the following algorithms:

1. Advanced Encryption Algorithm (AES),
2. Triple Data Encryption Algorithm (Triple-DES),
3. Data Encryption Algorithm (DES),
4. Digital Signature Algorithm (DSA),
5. Secure Hash Algorithm (SHA),
6. Random Number Generator (RNG),
7. Reversible Digital Signature Algorithm (RSA),
8. Elliptic Curve Digital Signature Algorithm (ECDSA),
9. Keyed-Hash Message Authentication Code (HMAC),
10. Counter with Cipher-Block Chaining-Message Authentication (CCM)
11. CMAC Algorithm (CMAC)

2 General Algorithm FAQs (Can be applied to all algorithms)

GEN.1 Where is the documentation for each algorithm validation system found?

Refer to the individual validation system guides for each supported algorithm for an explanation of the validation tests required for that specific algorithm. These validation guidelines are located in the specific algorithm section found on the csrc.nist.gov/cryptval website. For example, to find the Advanced Encryption Standard Algorithm Validation Suite (AESAVS), go to the csrc.nist.gov/cryptval website. Select Symmetric Keys in the blue column on the left. This page gives information for AES, TDES, DES, and Skipjack. A link to the AESAVS is in the Testing Requirements section. The individual validation guidelines for the currently supported algorithms are:

1. The Advanced Encryption Standard Algorithm Validation Suite (AESAVS),
2. NIST Special Publication 800-20, Modes of Operation Validation System for the Triple Data Encryption Algorithm (TMOVS): Requirements and Procedures. (An additional test, the Multi-block Message Text (MMT), is also required.),
3. NIST Special Publication 800-17, Modes of Operation Validation System (MOVS): Requirements and Procedures (The DES algorithm also requires the completion of the MMT tests.),
4. The Digital Signature Algorithm Validation System (DSAVS),
5. The Secure Hash Algorithm Validation System (SHAVS),
6. The Random Number Generator Validation System (RNGVS),
7. The Reversible Digital Signature Algorithm Validation System (RSAVS),
8. The Elliptic Curve Digital Signature Algorithm Validation System (ECDSAVS),
9. The Keyed-Hash Message Authentication Code Validation System (HMACVS),
10. The Counter with Cipher-Block Chaining-Message Authentication Validation System (CCMVS)
11. The CMAC Validation System (CMACVS)

GEN.2 Is it acceptable if an implementation of an algorithm is presented in such a manner that the end user using the implementation must make calls to several functions in order to perform a major function of the algorithm (for example, Signature Generation)?

Generally, no. NIST expects that all the parts of an implementation of an algorithm will be contained within one executable (or its equivalent in firmware or hardware) and that one call to the algorithm implementation will determine which of the underlying functions are executed, how these underlying functions are executed, and in what order

these functions are executed. For example, in a PKCS1.5 implementation, as the PKCS#1 v2.1 document states, we would expect that a call to RSASSA-PKCS1-V1_5_SIGN would call EMSA_PKCS1_V1_5_ENCODE and RSASP1. The CAVS testing has been designed to assure that the functionality of the underlying functions within an algorithm implementation is operating correctly. If all the parts are supplied to an end user with the ability to put them together any way possible, there is no guarantee that they will be called in the order specified by the standard for that algorithm. Therefore, we cannot validate this implementation as a completed implementation.

However, there are cases where two or more distinct entities in a system cooperate to execute an algorithm, such as the case of a smart card and a smart card reader. In this case the functions that comprise the digital signature algorithm are divided between the two parts of the system, card and reader, and the order of operations is fixed so that there is no way for the component functions of the algorithm to be called out of order.

GEN.3 How should the algorithm implementation be named?

There are no requirements on the algorithm implementation name. If the algorithm implementation is a part or component of a cryptographic module, it should not have the same name as the module itself. If the algorithm implementation is itself a module, then only one name is needed. There is no requirement to have algorithm names such as "AES" or "SHA" in the name of the implementation.

GEN.4 If an algorithm implementation performs more than one algorithm (for example, if an algorithm implementation named XYZ CryptoLib2000 performs both AES and SHA), can a different description be given for each algorithm?

No, the implementation description for an implementation applies to all algorithms implemented by this implementation. The same description will be displayed on all algorithm validation lists for this implementation. (In the example above, the same description will be displayed on the AES and SHA validation lists for this algorithm implementation.)

GEN.5 Are there prerequisites to having some algorithms validated?

Yes. The following list specifies a list of algorithm components that, if tested, need additional testing to be performed in other algorithm sections. For example, in DSA Key Generation testing, the underlying RNG engine is not stressed in a complete fashion;

therefore, the RNGVS suite must be performed on the underlying RNG engine in order to receive validation of the DSA implementation. .

Algorithm Tested		Additional Required Test(s)
AES		
	Counter mode	AES – ECB mode
TDES		
	Counter mode	TDES – ECB mode
DSAVS		
	Domain Param Gen	SHA-1[1]
	Domain Param Ver	SHA-1[2]
	Key Gen	RNG
	Sig Gen	SHA-1, RNG (because of per message secret #)
	Sig Ver	SHA-1
RNG		
	186 RNG	**DOES NOT REQUIRE PREREQUISITE TESTING** (Notes: Uses a SHA-like function. Therefore ECDSA RNG SHA does not need to be validated. The DES algorithm is tested sufficiently by the RNG for use by the RNG function.)
	ANSI X9.31	**DOES NOT REQUIRE PREREQUISITE TESTING** (Notes: The underlying algorithms are tested sufficiently by the RNG for use by the RNG function.)
RSA		
	KeyGen9.31 SigGen9.31 SigGenPKCS1.5 SigGenPSS SigVer9.31 SigVerPKCS1.5 SigGenPSS	**For only KeyGen9.31**: RNG **For all functions**: The supported SHA algorithm: SHA-1, SHA-256, SHA-384, or SHA-512 (SHA-224 for PKCS versions)

[1] Uses SHA-1, but this is not a "hidden" value as when generating the private key, *x*, or the per message secret value, *k*. If this process is done incorrectly, the correct value of Q cannot be determined.
[2] same as above.

HMAC	The supported SHA algorithm(s)
CCM	Some mode of AES used by the CCM
ECDSA	
Key Pair	RNG
PKV	Nothing
Sig Gen	SHA, RNG (because of per message secret #)
Sig Ver	SHA
CMAC	The underlying encryption algorithm(s) and mode(s) of operation and states (encryption and decryption) implemented in the CMAC implementation; i.e., the AES algorithm and/or the TDES algorithm
DRBG	
HASH_DRBG	SHA
HMAC_DRBG	HMAC
CTR_DRBG	NIST-Approved symmetric key algorithm using Counter mode. Currently, NIST approves both the AES and TDES algorithms for use with DRBG.
Dual EC_DRBG	ECDSA Key Generation function (to test the point multiplication function) SHA

GEN.6 An algorithm implementation has restrictions on it because of the application that contains it. Can I validate the algorithm implementation?

In order for a cryptographic algorithm to be validated, the algorithm must be designed in such a way as to allow for testing by the validation tests. It must also be designed as specified in the corresponding official algorithm document. If these two conditions are not met, the cryptographic algorithm implementation cannot be validated. If the restrictions of the application interfere in testing the algorithm or designing the algorithm according to the specifications in the standard, this algorithm cannot be validated.

GEN.7 Guidance on the relationship between the operating environment for cryptographic algorithm implementation

validations and the operating environment for cryptographic modules.

Implementation Guidance (IG) 1.4, Binding of Cryptographic Algorithm Validation Certificates, identifies the configuration control and operational environment requirements for the cryptographic algorithm implementation(s) embedded within a cryptographic module when the latter is undergoing testing for compliance to FIPS 140-2. This IG states:

> For a validated cryptographic algorithm implementation to be embedded within a software, firmware or hardware cryptographic module that undergoes testing for compliance to FIPS 140-2, the following requirements must be met:
>
> 1. the implementation of the validated cryptographic algorithm has not been modified upon integration into the cryptographic module undergoing testing; and
> 2. the operational environment under which the validated cryptographic algorithm implementation was tested by CAVS must be identical to the operational environment that the cryptographic module is being tested under by the CMT laboratory.

GEN.8 Guidance on what to do when the zip file being sent to NIST containing the CAVS results is too large to send via email.

Winzip has an option under the Action screen called Split. It will automatically make multiple files each a manageable size. Send each of the files indicating this is 1 out of X, this is 2 out of X, etc.

GEN.9 What should be done if an algorithm implementation is housed on two different version numbers of a chip?

Generally, any two implementations of an algorithm that have different version numbers must be validated separately regardless of the physical differences. Two sets of files must be generated by the CAVS tool to test both operating environments. However, the vendor is not required to choose a packaged IC as the physical boundary. The CAVP allows validation of a die, so if a die is validated for an algorithm or algorithms, then any packaged ICs that contain the die do not need to be validated again for the same algorithms.

GEN.10 Suppose an algorithm implementation has been validated on a chip. What happens when a change is made to this chip? . It is claimed that the change is not directly related to the algorithm implementation. Is the algorithm validation still valid?

The laboratory would need to determine whether or not this change modified the environment. If the environment was changed, the algorithm implementation needs to be revalidated.

GEN.11 If a vendor claims that their implementation runs on multiple operating systems, how should this be thoroughly validated?

The operating environment(s) listed for a validated cryptographic algorithm implementation tested under the CAVP indicates the operating environment(s) on which the implementation was tested.

A separate set of test vectors will be generated by the CAVS tool. The vendor will use a different set of test vectors to test each different supported operating environment. Each of these operating environments will be listed on the algorithm validation certificate and the website.

GEN.12 A vendor has indicated that the version number of a previously validated algorithm implementation has changed. They indicate that the version number change is not security relevant; none of the code within the algorithm implementation boundary has been changed. What should be done?

The laboratory would need to verify through source code review and documentation review that the version number change definitely is not a security relevant change and that none of the code within the algorithm boundary has been modified. An official change request would be submitted to NIST by the laboratory requesting a version number change indicating that the laboratory has verified that the change does not constitute a security relevant change.

GEN.13 What information is required in the Operational Environment field?

When submitting the algorithm test results to the CAVP, the operational environment on which the testing was performed must be specified (e.g. including modified module identification or simulation environment). [REF: FIPS 140-2 IG G.11 and 1.4]

For Software implementations, the following information must be listed:

1. Processor – This field shall identify the vendor and processor family. Processor speed is not required (retesting of an implementation is not required for different processor speeds).

   Examples of correct processor entries (this is not a complete list):

   a. AMD
      - AMD Athlon™
      - AMD Athlon™ XP
      - AMD Athlon™ 64
      - AMD-Opteron®
      - AMD Turion™ 64
      - AMD Sempron™
      - Am486®
      - AMD-K5®
      - AMD-K6®
      - AMD-K5® III
      - AMD-Phenom® X3
      - AMD-Phenom® X4

   b. Intel
      - Intel486™
      - Intel® Celeron®
      - Intel® Celeron® D
      - Intel® Itanium®
      - Intel® Itanium® 2
      - Intel® Xeon™
      - Intel® Pentium
      - Intel® Pentium® II
      - Intel® Pentium® II Xeon™
      - Intel® Pentium® III
      - Intel® Pentium® III Xeon™
      - Intel® Pentium® 4
      - Intel® Pentium® M
      - Intel® Pentium® Pro
      - Intel® Xeon®
      - Intel® Core™2
      - Intel® Core™2 Duo
      - Intel® Core™2 Quad
      - Motorola
      - Motorola PowerPC® MPC7410 G4
      - Motorola PowerPC® MPC7455
      - Motorola PowerPC® 750GX
      - Motorola PowerPC® MPC7457
      -

   c. Sun
      - Sun UltraSPARC® T1
      - Sun UltraSPARC® IV+
      - Sun UltraSPARC® IV
      - Sun UltraSPARC® III
      - Sun UltraSPARC® IIIi
      - Sun UltraSPARC® IIi

2. Operating system – This field shall identify the vendor and operating system family.

Examples of correct operating system entries (this is not a complete list):

a. Microsoft
   - Microsoft Windows 95
   - Microsoft Windows 98
   - Microsoft Windows 2000
   - Microsoft Windows Server 2003
   - Microsoft Windows CE
   - Microsoft Windows ME
   - Microsoft Windows NT
   - Microsoft Windows XP
   - Microsoft Windows Vista

b. Sun
   - Sun Solaris 10
   - Sun Red Hat Enterprise Linux
   - Sun SuSE Linux Enterprise Server
   - Sun Solaris 10 with SUN JRE 5.0

c. IBM
   - IBM AIX 5L™
   - IBM z/OS
   - IBM OS/390
   - IBM MVS
   - IBM z/VM
   - IBM VM/ESA
   - IBM z/VSE

d. Wind River
   - WindRiver VxWorks 6.x

e. FSMLabs
   - FSMLabs RTLinux

For Firmware implementations, the following information must be listed:

1. Processor: This field shall identify the vendor and processor family. Processor speed is not required (retesting of an implementation is not required for different processor speeds). For examples, refer to the Software Processor section above.)

   The following examples are not acceptable entries for this field because they are general terms and don't uniquely identify a processor:

   a. DSP Processor
   b. FPGA

For Hardware implementations, the environment is the actual hardware device. Therefore, N/A is indicated in the Operating Environment since the implementation name and hardware part number indicate the environment on which the IUT was run.

If a cryptographic algorithm implementation can not be tested in its hardware environment, per IG G11, a simulator may be used to test the algorithm implementation. The algorithm implementation would be extracted from the hardware implementation and tested with a simulator. In this case, the implementation would be firmware and the operating environment would list the name of the simulator used to test the implementation. Example of simulator names:

a. Simulator:
   - Aldec
   - Active HDL version 6.1

GEN.14 A vendor implements an algorithm that requires prerequisite algorithm validations. The prerequisite algorithm implementation used by the vendor is housed in a validated cryptographic module. The algorithm was not validated because, at the time the module was tested, validation testing for this algorithm did not exist. Can the vendor use this algorithm implementation as a prerequisite algorithm?

Yes, this implementation of the prerequisite algorithm can be used by another cryptographic algorithm. Because validation testing for this algorithm did not exist when its module was validated, the cryptographic algorithms within this module validation will be grandfathered in. The cover letter should include explanation of this and the certificate number for the underlying algorithm should list the 140-1 Module Validation Certificate number.

For example, a vendor implements DSA. As a prerequisite, RNG needs to be validated. The RNG implementation this vendor is using is in a module that was validated prior to there being RNG testing. It can be used in the DSA implementation. The cover letter will indicate that the implementation of RNG used by the DSA is grandfathered in because it was tested under FIPS 140-1. The RNG certificate number listed will be the 140-1 Module Certificate number.

3 AES FAQ

AES.1 Guidance on evaluating the Counter Mode (for either the AES or TDES algorithm) if it is implemented in hardware.

When a CMT Laboratory contracts with a vendor for testing regarding the documentation and specification of the cryptographic module, the documentation requirements in Section 4.1 and Section 4.10.3 are applicable. The vendor must supply sufficient documentation for either cryptographic module testing to FIPS 140-2 or for algorithm testing (e.g. CCM mode) to the CMT Lab. Module or algorithm validation cannot occur if a CMT Laboratory cannot demonstrate access to such documentation.

As applied to this specific question, to evaluate an AES counter mode implemented in hardware, the documentation, the HDL (which is equivalent to the source code for software), and the hardware schematics of this hardware implementation would be needed. Documentation alone is not enough because there is no way to prove it is an accurate account of how the hardware actually works. The HDL and the schematics are also needed to provide the necessary information to thoroughly evaluate the counter mode and to prove the documentation is correct.

AES.2 What is required to get an AES or TDES Counter Mode implementation validated?

To validate an AES or TDES Counter Mode implementation in software, firmware, and/or hardware, an examination of the processing and the logic involved in this implementation shall be performed. The lab shall send a detailed documented review of their examination to CAVP. If applicable, the review shall include an assessment of the source code and/or the executable. This supplied documentation shall be sufficient to provide evidence that the implementation conforms to the standard requirements.

In addition to the above requirements, to validate a hardware implementation of AES or TDES Counter Mode, the review shall include an assessment of the HDL (which is equivalent to the source code for software), and the hardware schematics of the hardware implementation.

If the implementation being tested derives counter values internally, the testing lab shall conduct a design review of the internal counter mechanism used to ensure that it provides unique counter block values as well as a code review of the implementation of the counter mechanism to ensure that it properly implements the design. An assessment of this information shall be included in the supplied documentation.

NOTE: The prerequisite for validating an AES or TDES Counter Mode implementation is to successfully perform the validation of the ECB mode of operation on the encryption engine used for the Counter Mode.

4 DES FAQ

NOTE: The CAVP has discontinued the issuance of new DES algorithm validation certificates as of February 9, 2005. DES implementations under contract with a CMT laboratory prior to February 9, 2005, will be completed. See the DES Transition Plan for more details.

5 Triple-DES FAQ

TDES.1 Guidance on evaluating the Counter Mode (for either the AES or TDES algorithm) if it is implemented in hardware.

When a CMT Laboratory contracts with a vendor for testing regarding the documentation and specification of the cryptographic module, the documentation requirements in Section 4.1 and Section 4.10.3 are applicable. The vendor must supply sufficient documentation for either cryptographic module testing to FIPS 140-2 or for algorithm testing (e.g. CCM mode) to the CMT Lab. Module or algorithm validation cannot occur if a CMT Laboratory cannot demonstrate access to such documentation.

As applied to this specific question, to evaluate a TDES counter mode implemented in hardware, the documentation, the HDL (which is equivalent to the source code for software), and the hardware schematics of this hardware implementation would be needed. Documentation alone is not enough because there is no way to prove it is an accurate account of how the hardware actually works. The HDL and the schematics are also needed to provide the necessary information to thoroughly evaluate the counter mode and to prove the documentation is correct.

6 DSA FAQ

DSA.1 If vendors are having problems getting their PQGGen or SigGen to work properly, where can a known set of values be obtained to help in their testing?

Vendors having problems with a PQGGen or SigGen should generate a sample of values by running the PQGVer or SigVer test and extracting one of the groups of data that **pass (P).** If the implementation being tested does not come up with the correct signature, then the vendor may assume that there is something wrong with the implementation.

DSA.2 In the X186 RNG, does an implementation have to support the optional seed XSEED?

An implementation does not have to support the optional seed.
In the DSA algorithm, the XSEED value is added to the XKEY value. This value is then moded 2^b to compute the XVAL. In the CAVS tool, the SEED value is set to 0 because it is only used in an addition function and the purpose of the test is not to check if the implementation can add two numbers together. Instead the purpose of this validation test is to assure that the implementation performs the G function correctly.

DSA.3 When implementing the RSA key generation algorithm according to ANSI X9.31, Digital Signatures Using Reversible Public Key Cryptography, is it acceptable to generate primes using the procedure detailed in Appendix E.4 of the ANSI X9.31 standard instead of the procedure described in section 4.1.2.1 of the same standard. Moreover, if this is acceptable, what sort of primality testing needs to be done? Appendix E.4 is not very clear in this respect.

Appendix E.4 can be used in addition to, but not in place of Section 4.1.2.1.
Appendix E.4 contains the same calculations for generating the private prime factors that are found in Section 4.1.2.1. The only difference is that Appendix E.4 explains how to find the first prime after the first random X is selected by using sieving; it informs the implementer how to do this. Section 4.1.2.1 does not specify how to select this value. Therefore, one could add this processing to an implementation.

Appendix E.4 does not specify how to do the primality testing of Y. But since this is a very important step, it is specified in Section 4.1.2.1. Therefore, it is important that this part of Section 4.1.2.1 is performed.
Because of these requirements, the informative method described in Appendix E.4 cannot be substituted for the method described in Section 4.1.2.1. However, it can be used in addition to Section 4.1.2.1.

Currently, ANSI X9.31 is being updated by American Standards Committee (ASC) X9, Financial Services. RSA Security is the editor of ANS X9.31 within ASC X9; the updated version may allow certain alternative primality tests if they provide an equivalent threshold of assurance, as specified in ANS X9.80 Prime Number Generation Primality Testing and Primality.

DSA.4 From DSA.3 above, it seems that the procedure outlined in Appendix E.4 simply provides a fast method of generating the values for p1, p2, q1, and q2 from their respective X values. As a result, the process of generating p and q from these values must follow Section 4.1.2.1. Can you confirm this?

Also, would the RSA key generation algorithm testing be affected if a vendor chooses to use Appendix E.4 to generate p1, p2, q1 and q2? Appendix E.4 mentions that the sieving method will remove substantial composite numbers as well as small primes; however, section 4.1.2.1 mentions that p1, p2, q1, and q2 are the FIRST primes greater than their respective X values. Since using the sieving method results in some of the smaller primes being sieved out, is it possible that the values of p1, p2, q1, and q2 obtained using the sieving method of E.4 will be different from those values expected by using section 4.1.2.1? If the values for p1, p2, q1 or q2 are different, the resulting p and/or q will be different from what is expected by the algorithm test tool. Will using E.4 affect the key generation algorithm testing?

Yes, it can be confirmed that the generation of p and q must follow Section 4.1.2.1; in particular, they must be the first primes after the respective randomly generated values that satisfy all of the properties listed in that section, including passing the 8 rounds of the Miller-Rabin test followed by the Lucas test. But that does not preclude sieving the candidate values of p and q as described in Annex E.4, similar to the sieving of the candidate values for p1, p2, q1, and q2.
The sieving process should not remove any candidate primes. Because the sieving primes are all much smaller than the candidate primes, the sieving process should remove *only* composites, i.e., non-trivial multiples of the sieving primes.

Actually, the opposite problem is theoretically possible, namely, that the probabilistic primality test in Section 4.1.2.1 will identify some number as prime that the sieving method in Annex E.4 eliminates as composite. But the same discrepancy is also theoretically possible for two different implementations of the probabilistic primality test

in Section 4.1.2.1; e.g., using different sets of bases for the Miller-Rabin test. The probability of such an event in practice, however, is sufficiently small for us to discount it.

The sieving in Annex E.4 should not affect validation testing, assuming that it is implemented correctly, of course, and that the remaining candidates are properly tested for primality. The validation testing does not directly exercise the sieving process, but, as discussed above, whether or not the sieving process is used, the same answer should be the achieved with overwhelming probability.

7 SHA FAQ

SHA.1 A vendor wants to test SHA-1 (byte only). However, nothing is hashed greater than 256 bytes. How is this implementation validated?

This is similar to the situation that occurs when a hash implementation does not handle the NULL string. As in that case, the CMT Lab will generate values using the CAVS tool. Only the values supported by the restrictions of the implementation will be used in the validation of the implementation. The CMT Lab should check the response files individually to assure that the messages satisfying the restriction pass successfully. Also, on the algorithm submission in the cover letter (and email request), the CMT Lab will indicate the special case and will explain how the files were verified. The restriction will be indicated on the algorithm validation certificate and on the algorithm validation list website.

8 RNG FAQ

RNG.1 When generating RNG test vectors for the General Purpose RNG, both the Xorg and Korg generators were selected. Values for Korg were not generated for General Purpose RNG. Why?

This confusion is caused by adding the General Purpose RNG to an existing screen in the CAVS tool. The original RNG uses Xorg, Xchange, Korg and Kchange. But the General Purpose RNG, as specified in the standard, only uses Xorg and Xchange.

Because of the sharing of this screen in the CAVS tool, if Korg and Kchange are selected for General Purpose RNG, they are ignored. If the original RNG and General Purpose RNG are selected and Korg, Kchange, Xorg and Xchange are selected, the tests for the original RNG using Korg, Kchange, Xorg and Xchange will be generated as well as the tests for the General Purpose RNG using Xorg and Xchange.

In a later release of the CAVS tool, a separate screen will be developed to clarify this situation.

RNG.2 A vendor implementing the algorithm in Appendix 3.1 eliminated step 3d which calculates a new XKEY. Instead, a new random XKEY was created. Is this acceptable?

By eliminating step 3d from the implementation, the algorithm is not implemented according to the specifications in the standard. An algorithm must be implemented according to the specifications in the associated standard in order to be recognized as a NIST-Approved algorithm.

9 RSA FAQ

RSA.1 If a vendor is having problems getting the GenKey or SigGen to work properly, where can a known set of values be obtained to help in the testing?

If a vendor is having problems with a SigGen, a sample of values can be generated by running the SigVer test and extracting one of the groups of data that **pass (P).** If the implementation being tested does not come up with the correct signature, then it can be concluded that there is something wrong with the vendor's implementation.

RSA.2 What should be done in the situation where a vendor supports a different salt length and value for each SHA algorithm supported in RSASSA-PSS? Currently, only one salt value can be specified on the CAVS screen. This salt value is then applied to all SHAs selected.

A modification will be made to the CAVS tool for RSASSA-PSS to allow the entry of different salt lengths (and salt values, if applicable) per SHA algorithm/mod size. This will be changed in a future release of the CAVS tool. This will allow for different salt lengths for each SHA algorithm specified.
Until then, a new project folder will have to be generated for each SHA/salt combination.

RSA.3 When generating RSASSA-PSS in the Signature Verification screen for the SHA-512 implementation with a salt length of 64 bytes and all mod sizes, CAVS (version 4.3) returned a "Fatal Error" message and indicated that the vectors were generated but with errors. Why did this happen?

According to the specifications in the PKCS#1 v2.1 document, this error should be returned when the following condition occurs (See pg 35, 9.1.1 Encoding operation - EMSA-PSS-ENCODE):

if emLen < hLen +sLen + 2, output "encoding error" and stop.

Explanation:
The emLen = length of the encoded message = modulus size. In the files, the first mod size selected was 1024.

In this case (which is the case that fails), emLen =modsize = 1024bits = 128bytes
The hash length (hLen) is 512bits = 64 bytes
The salt length (sLen) is 64 bytes.
Therefore, 128 < 64+64+2 is true causing the "encoding error" message to be output.

The only guidance on salt sizes in the PKCS document on page 34 states that "**Typical salt lengths in octets are *hLen* (the length of the output of the hash function Hash) and 0.** ...."

In this situation, where mod = 128 bytes and hash = 64 bytes, the salt length can not be 64 bytes without making this formula fail. In reality, the 1024 mod size was not intended to be used with SHA-512.

NIST has not published any guidance on the interoperability of mod sizes, hash functions, and salt lengths.
But we have drafted a proposed change to FIPS 186-2, Digital Signature Standard, as follows:

nlen bits	*emLen (bytes)*	*hash function*	*outlen (bits)*	*hLen (bytes)*	*max sLen (bits/bytes)*	
1024	128	SHA-1	160	20	848 bits = 106 bytes	128 >= 20+2+106
2048	256	SHA-224	224	28	1808 bits = 226 bytes	256 >= 28+2+226
2048	256	SHA-256	256	32	1776 bits = 222 bytes	256 >= 32 + 2 + 222
3072	384	SHA-256	256	32	2800 bits = 350 bytes	384 >= 32 + 2 + 350

This new guidance should help with the issue raised in this question (RSA.3). Note that this is proposed guidance may be modified before it is issued in final form.

Also, in the future the CAVS screen will be redesigned to allow for different salt lengths (and values) for each mod size and SHA. Until then, if more than one mod size is selected, the CAVS tool would have to be run separately for each mod size to avoid this problem.

RSA.4 In the SigVerX.fax files, what does the number in parentheses after the result =F field mean?

The number indicates what value was changed to make the signature fail.

(1) Message was changed
(2) Public Key was changed
(3) Signature was changed

RSA.5 Is it acceptable to generate primes using the procedure detailed in Appendix E.4 of the ANSI X9.31 standard

instead of that described in section 4.1.2.1 of the same standard? Moreover, if this is acceptable, what sort of primality testing needs to be done? Appendix E.4 is not very clear in this respect.

The CAVP compared Appendix E.4 of the ANSI X9.31 standard with Section 4.1.2.1 of the same standard to determine of one could be substituted for the other. We concluded that Appendix E.4 can be used in addition to, but not in place of Section 4.1.2.1.

Appendix E.4 contains the same calculations for generating the private prime factors that are found in Section 4.1.2.1. The only difference is that Appendix E.4 explains how to find the first prime after the first random X is selected by using sieving; it informs the implementer how to do this. Section 4.1.2.1 does not specify how to select this value. Therefore, one could add this processing to an implementation.

Appendix E.4 does not specify how to do the primality testing of Y. But since this is a very important step, it is specified in Section 4.1.2.1. Therefore, it is important that this part of Section 4.1.2.1 is performed.

Because of these requirements, the informative method described in Appendix E.4 cannot be substituted for the method described in Section 4.1.2.1. However, it can be used in addition to Section 4.1.2.1.

Currently, ANSI X9.31 is being updated by American Standards Committee (ASC) X9, Financial Services. RSA Security is the editor of ANSI X9.31 within ASC X9; the updated version may allow certain alternative primality tests if they provide an equivalent threshold of assurance, as specified in ANSI X9.80, Prime Number Generation, Primality Testing and Primality.

RSA.6 The Key Generation function of an RSA implementation does not output d, the private signature exponent. This is valid according to ANSI X9.31 Section 4.1 where the outputs from key generation are listed. How do I test this IUT?

Both the Key Generation test and the Signature Generation test must be run to test an IUT that implements RSA Key Generation where d is not output. The Signature generation function may exist inside or outside the IUT, depending on whether or not the IUT implements signature generation. If the IUT does not provide signature generation, this function must be obtained outside the IUT as part of the test code. If it does provide signature generation, then this function is part of the IUT. In either case, the keys used by the Signature Generation test **must be generated by the IUT's 9.31 Key Generation**

implementation.

The Key Generation test will provide proof that the p, q and n values are generated correctly. The d value will be invalid since it can not be output by the IUT.

The Signature Generation test will provide proof that the keys generated by the IUT can be used successfully by the IUT to sign messages using d (Signature = min { RR ^ d mod n, n - (RR ^ d mod n)}) and by CAVS to verify the signature using e. If the signature fails, the IUT does not pass.

When submitting the IUT to be validated, an explanation of the testing process should be included.

10 HMAC FAQ

HMAC.1 If an implementation supports other MAC size than those supported by the CAVS tool, how are these MACs tested?

The CAVP cannot test every MAC size. Instead, several MAC sizes throughout the valid range have been selected for testing. At least one of the specified MAC sizes must be supported by the implementation.

All values on the HMACVS and the CAVS for HMAC are dealing with values in BYTES. Therefore all values are AUTOMATICALLY divisible by 8 (since 1 byte = 8 bits).

HMAC.2 An implementation supports all 3 ranges of values (K<B, K>B, and K=B). Does this mean that 3 separate tests should be run for the same implementation or will the CAVS tool allow us to choose all 3 ranges?

The CAVS tool will allow for all three ranges to be selected at the same time.
Enter 2 length values for K<B, 2 length values for K>B and check the K=B box.
All these length values will be used in the data that is produced.

HMAC.3 An implementation only supports one K length size < B. How should this be indicated since the CAVS tool requires the entry of two values of K < B to be tested?

The CAVS tool requires that two values of K<B to be supplied to provide more testing for the implementation. But in the case where only one value is supported by the implementation, simply enter the same value for K<B in both places. The tool will generate the request file with two sets of data to test the key size allowed.

The same process is applicable to K>B.

HMAC.4 If an HMAC implementation uses a SHA implementation that cannot be tested separately, does the SHA algorithm have to be tested? Why?

When an implementation of the HMAC algorithm is validated, the CAVP requires that the SHA algorithm has been previously validated. Even though the HMAC algorithm relies on the correctness of the SHA algorithm, the HMAC testing alone does not provide for adequate testing of the SHA algorithm. The HMAC tests focus on testing the HMAC processing only.

The CAVP requires additional "stress testing" of the underlying SHA algorithm which is provided in the SHA Validation tests.

This requirement cannot be bypassed.

11 CCM FAQ

CCM.1 A hardware implementation of AES CCM has been developed to be used for IEEE 802.11i communications. The CCM implementation cannot perform the validation tests because of restrictions as specified in 802.11i. Can the CCM implementation be validated?

To validate the CCM algorithm, the algorithm must be designed in such a way as to allow for it to be tested. It must also be designed as specified in IEEE the official CCM document. If these two conditions are not met, the CCM implementation can not be validated. Any restrictions put on the algorithm as a result of the IEEE 802.11i protocol is outside the scope of the CCM algorithm validation testing.

CCM.2 If a CCM implementation only supports specific lengths for the Associate Data field because of IEEE 802.11i restrictions, can it be validated?

If a CCM algorithm validation only supports specific byte lengths for the Associate Data field, a special note would be included on the validation list and the certificate indicating the restriction that only those supported lengths were validated. The fact that the restriction is associated with the IEEE 802.11i protocol is irrelevant.

CCM.3 Is it possible for an implementation to implement encrypt only or decrypt and verify only? Is it possible to then test only that one function?

Yes it is possible for a CCM implementation to only implement the encrypt function or the decrypt and verify function. In this situation, only that function would be validated and the algorithm validation certificate will indicate this information. If the implementation only supports encrypt, the variable associated adata test, the variable payload test, the variable nonce test, and the variable tag test will be required to validate the implementation. If the implementation only supports decrypt and verify, the decryption-verification process test will be required to validate the implementation. Currently, the laboratory generates all 5 files at the same time. The lab would then only forward the appropriate request and sample files to the vendor for testing.

12 ECDSA FAQ

ECDSA.1 For ECDSA PKV validation testing, why are values for X and Y different lengths? As the question states: "…our client's ECDSA implementation is having some problems with two of the curves located in PKV. To the best of my knowledge for B-163 and K-163, both proper values for Qx and Qy are supposed to be 21 bytes in length. However, it appears the CAVS tool expects values that are 22 bytes in length to be valid Public Keys. For example, the CAVS tool expects the first iteration for K-163 to be a valid Public Key but their implementation determines it to be invalid.

Qx = 0001c59c158ff8b2f8d113542922e6952ea8dcd88c89
Qy = 00059f2d1622c0c89edd9ffc6901eadc31b42050cc44

Again, it appears that values for the above iteration are 22 bytes in length, which is why their implementation determines the public key is invalid.

Here is another example from K-163:

Qx = 34696e17431234b071bbb9674cc1f8b82a7ebd52
Qy = 0007788cd02faaa8289d1f61bd7db262572cf870a783

The Qx is 20 bytes in length and Qy appears to be 22 bytes in length, which their implementation determines to be invalid when the CAVS tool expects it to be a valid public key."

All values should be thought of as numbers. All valid values have a MAXIMUM value of 21 bytes (163 bits = 21 bytes (really 20 bytes and 1 nibble)). One of the tests for PKV validation checks to see if the implementation can identify when a value is out of range – that is greater than 21 bytes. That is why the file indicates a 22 byte number for Qx and Qy. IT IS NOT A VALID NUMBER AND SHOULD FAIL.

In the PKV files, if a value is less than 21 bytes in length (e.g., 20 bytes), only the 20 bytes are printed; i.e., the zeros are added only if there is a partial byte. That is why the file looks as if some values are longer than others. But if every value in this file is read as a number, it will be read in correctly.

ECDSA.2 In the PKVVer.fax files, what does the number in parentheses after the result =F field mean?

The number indicates what value was changed to make the signature fail.
For the prime curve:

(1) Make Q_x or Q_Y out of range
(2) Point not on curve

For a Poly Curve:

(1) Point not on curve
(2) Add PT of order 2

ECDSA.3 Can an ECDSA implementation be validated if it does not use any NIST-recommended curves?

No. In order to validate an implementation of ECDSA, the algorithm implementation must implement at least one NIST-recommended curve. It can have non-recommended NIST curves as well as long as there is at least one NIST-recommended curve.

Other facts concerning cryptographic modules using ECDSA algorithm implementations:

1. All FIPS 140-2 validated modules (that implement ECDSA for use in the FIPS mode) must have an ECDSA algorithm certificate.
2. In order to receive an ECDSA algorithm (FIPS 186-2) certificate, the module must be tested using one of the NIST recommended curves.
3. A FIPS 140-2 module may use non-recommended NIST curves in the FIPS Approved mode of operation, if the module has successfully received an algorithm certificate.
4. The module itself (without modification) must implement and support testing of the ECDSA algorithm with a NIST-recommended curve. The validated modules boundary as specified by the provided version/PN/etc must support and have the ability to perform ECDSA with a NIST-recommended curve. It cannot be provided temporarily for testing in an emulator/simulator and then be removed from the "real" module.
5. If a vendor's module cannot support algorithm testing by using a NIST recommended curve, the ECDSA services of this module will be considered non-compliant.

13 CMAC FAQ

CMAC.1 What should be done in the situation where an implementation only supports one message length for either case where the message length is divisible by the Blocksize or the message length is not divisible by the Blocksize?

If the implementation only supports one message length that is divisible by the Blocksize, enter this length in both fields. The same applies to the situation where an implementation only supports one message length that is not divisible by the Blocksize.