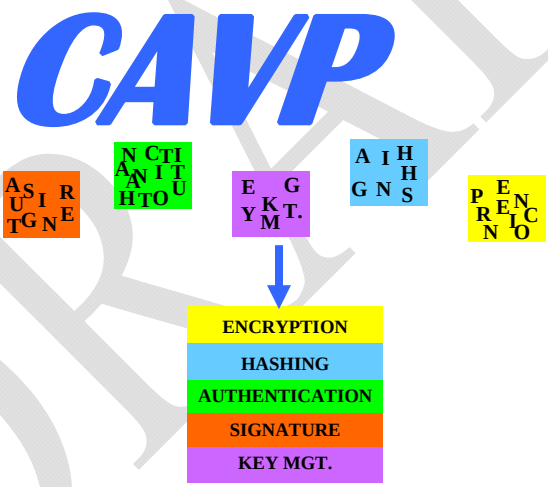
Frequently Asked Questions
For the
Cryptographic Algorithm Validation Program
Concerning the Validation of Cryptographic Algorithm Implementations

National Institute of Standards and Technology
Communications Security Establishment of Canada



Initial Release: May 16, 2005

Previous Update: November 30, 2011

Last Update: March 27, 2012

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

New FAQ's

GEN.21 Can a vendor request that an algorithm implementation be validated but not posted on the validation list until a later date?

GEN.22 Suppose you have two implementations that you are testing at the same time and each have their own separate cryptographic algorithm boundaries. Each is a prerequisite of the other. How do you get the validated since neither is validated at the time you are testing the other? For example,
You have the following implementations that have their own separate cryptographic boundaries:
1. An implementation of DRBG
2. An implementation of ECDSA

The prerequisite for DRBG is ECDSA and the prerequisite for ECDSA is DRBG. If the implementation of DRBG and ECDSA are using each other and neither is validated yet, how do you get them validated since each one is a prerequisite of the other?

DRBG.4 We validated our DRBG implementation before NIST SP 800-90A came out. Do we have to do a new validation?

Modified FAQ's

GEN.5 Added prerequisites for AES-XTS, SP 800-108 KDFS, and SP 800-135 KDFs.

DRBG.3 Added discussion of CTR_DRBG_Update function.

1 Introduction

Below is a compilation of questions received from the Cryptographic Security Testing (CST) laboratories relating to the validation of cryptographic algorithm implementations.

This is intended for use by the CST laboratories when validating cryptographic algorithms submitted by vendors. Vendors may find the information useful when developing implementations and working with the CST 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. Digital Signature Algorithm (FIPS186-2 and (FIPS186-3 DSA),
4. Secure Hash Algorithm (SHA),
5. Random Number Generator (RNG),
6. Reversible Digital Signature Algorithm ((FIPS186-2 and (FIPS186-3 RSA),
7. Elliptic Curve Digital Signature Algorithm ((FIPS186-2 and (FIPS186-3 ECDSA),
8. Keyed-Hash Message Authentication Code (HMAC),
9. Counter with Cipher-Block Chaining-Message Authentication (CCM)
10. CMAC Algorithm (CMAC)
11. Deterministic Random Bit Generator (DRBG)
12. Galois Counter Mode (GCM) and GMAC Algorithm
13. Key Agreement Schemes and Key Confirmation (NIST SP 800-56A) (KAS)
14. The XTS-AES Mode for Confidentiality on Storage Devices (XTS)
15. Component testing for
    a. The NIST SP 800-56A Elliptic Curve Cryptography Cofactor Diffie-Hellman (ECC CDH) Primitive
    b. The testing of “All of NIST SP 800-56A EXCEPT the KDF”

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 on the main page of the CAVP website. For example, to find the Galois Counter Mode (GCM) Validation Suite (GCMVS) on this page, select MAC - includes CMAC, CCM, GCM/GMAC, HMAC in the blue column on the left. GCM/GMAC is the third algorithm in this section. A link to the GCMVS is in the Testing Requirements section.

The individual validation guidelines for the currently supported algorithms and testable algorithm components 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. The Digital Signature Algorithm Validation System for FIPS 186-2 (DSAVS) and FIPS 186-3 (DSA2VS),
4. The Secure Hash Algorithm Validation System (SHAVS),
5. The Random Number Generator Validation System (RNGVS),
6. The Reversible Digital Signature Algorithm Validation System for FIPS 186-2 (RSAVS) and FIPS 186-3 (RSA2VS),
7. The Elliptic Curve Digital Signature Algorithm Validation System for FIPS 186-2 (ECDSAVS) and FIPS 186-3 (ECDSA2VS),
8. The Keyed-Hash Message Authentication Code Validation System (HMACVS),
9. The Counter with Cipher-Block Chaining-Message Authentication Validation System (CCMVS)
10. The CMAC Validation System (CMACVS)
11. The Deterministic Random Bit Generator (DRBG) Validation System (DRBGVS)
12. The Galois/Counter Mode (GCM) and GMAC Validation System (GCMVS)
13. The Key Agreement Scheme (KAS) Validation System (KASVS) (Also includes instructions for testing implementations of “All of SP800-56A EXCEPT KDF”.)
14. The XTS-AES Mode for Confidentiality on Storage Devices (XTS) (XTSVS)
15. The Elliptic Curve Cryptography Cofactor Diffie-Hellman (ECC CDH) Primitive Validation System (ECC_CDHVS).

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.

The testing of individual algorithm components was introduced in 2011. Several situations have led to this new type of testing. For example, PIV cards have limited processing space and therefore some parts of the algorithm are performed off card. Another example involves SP 800-56A which allows for the use of other KDFs not specified in 56A. In this situation, validation testing of "All of 56A except KDF" is performed.

(See GEN.18 for a related question.)

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. Following is an explanation of why. Some cryptographic algorithms make use of other cryptographic algorithms. For example, DSA Key Generation utilizes a random number generator. The algorithm validation test suites for each algorithm are designed to test the algorithm specifications, components, features, and/or functionality of that algorithm. So the validation tests for DSA Key Generation thoroughly tests the Key Generation function. But it doesn't thoroughly test calls to supporting cryptographic algorithms like the random number generator. Therefore, the random number generator validation tests need to be performed as a prerequisite to the DSA Key Generation validation testing to provide this assurance for the random number generator.

For algorithms that require an Approved Random Number or Bit Generator, any of the algorithms listed in FIPS 140-2, Annex C, may be used. The CAVP uses DRBG to refer to Approved Deterministic Random Number Generators specified in NIST SP 800-90A and RNG to refer to the rest of the Approved Deterministic Random Number Generators in Annex C. Unless explicitly stated otherwise, either RNG or DRBG may be used.

Algorithm Tested		**Additional Required Test(s)**
AES – Counter Mode		AES using any mode of operation that utilizes the forward cipher function.
CCM		AES using any mode of operation that utilizes the forward cipher function.
CMAC		NIST-Approved symmetric key algorithm (i.e., AES or TDES) using any mode of operation that utilizes the forward cipher function.
DRBG	HASH_DRBG	SHA

<table>
<tr><th colspan="2">Algorithm Tested</th><th>Additional Required Test(s)</th></tr>
<tr><td rowspan="4"></td><td>HMAC_DRBG</td><td>HMAC</td></tr>
<tr><td>CTR_DRBG</td><td>NIST-Approved symmetric key algorithm (i.e., AES or TDES) using any mode of operation that utilizes the forward cipher function.</td></tr>
<tr><td rowspan="2">Dual EC_DRBG</td><td>ECDSA Key Generation function (to test the point multiplication function)</td></tr>
<tr><td>SHA</td></tr>
<tr><td rowspan="5">DSA
(FIPS 186-2 or FIPS 186-3)</td><td>Domain Param Gen</td><td>SHA[1]</td></tr>
<tr><td>Domain Param Ver</td><td>SHA[2]</td></tr>
<tr><td>Key Gen</td><td>RNG or DRBG</td></tr>
<tr><td rowspan="2">Sig Gen</td><td>SHA</td></tr>
<tr><td>RNG or DRBG (because of per message secret #)</td></tr>
<tr><td>Sig Ver</td><td>SHA</td></tr>
<tr><td rowspan="4">ECDSA
(FIPS 186-2 or FIPS 186-3)</td><td>Key Pair</td><td>RNG or DRBG</td></tr>
<tr><td>PKV</td><td>None</td></tr>
<tr><td rowspan="2">Sig Gen</td><td>SHA</td></tr>
<tr><td>RNG or DRBG
(because of per message secret #)</td></tr>
<tr><td>Sig Ver</td><td>SHA</td></tr>
<tr><td colspan="2" rowspan="2">GCM</td><td>AES using any mode of operation that utilizes the forward cipher function.</td></tr>
<tr><td>RNG or DRBG, only if IVs generated internally</td></tr>
<tr><td colspan="2">HMAC</td><td>The supported SHA algorithm(s)</td></tr>
<tr><td>KAS (Key Agreement Scheme)</td><td>FFC (Finite Field Cryptography)</td><td>Underlying DSA algorithm.
The table below lists the DSA prerequisite function required based on the underlying supported cryptographic function(s):
<table>
<tr><th>Supported Cryptographic Function</th><th>DSA Prerequisite</th></tr>
</table>
</td></tr>
</table>

[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.

<table>
<tr><th colspan="2"><u>Algorithm Tested</u></th><th><u>Additional Required Test(s)</u></th></tr>
<tr><td colspan="2"></td><td><table>
<tr><td>Domain Parameter Generation</td><td>PQG Generation; PQG Verification</td></tr>
<tr><td>Domain Parameter Verification</td><td>PQG Verification</td></tr>
<tr><td>Key Pair Generation</td><td>Key Pair</td></tr>
<tr><td>Key Regeneration</td><td>Key Pair</td></tr>
</table></td></tr>
<tr><td colspan="2"></td><td>SHA</td></tr>
<tr><td colspan="2"></td><td>RNG or DRBG</td></tr>
<tr><td></td><td></td><td>If Key confirmation is being tested, all applicable MACs:
CCM
CMAC
HMAC</td></tr>
<tr><td></td><td>ECC (Elliptic Curve Cryptography)</td><td>Underlying ECDSA algorithm
The table below lists the ECDSA prerequisite function required based on underlying supported cryptographic function(s):<table>
<tr><th>Supported Cryptographic Function</th><th>ECDSA Prerequisite</th></tr>
<tr><td>Full Public Key Validation</td><td>PKV</td></tr>
<tr><td>Key Pair Generation</td><td>Key Pair</td></tr>
<tr><td>Key Regeneration</td><td>Key Pair; PKV</td></tr>
</table></td></tr>
<tr><td colspan="2"></td><td>SHA</td></tr>
<tr><td colspan="2"></td><td>RNG or DRBG</td></tr>
<tr><td></td><td></td><td>If Key confirmation is being tested, all applicable MACs:
CCM
CMAC
HMAC</td></tr>
<tr><td>KDF
(NIST SP 800-108)</td><td>Algorithm used to generate the key derivation key</td><td>SP800-56A KAS and/or
SP800-90A DRBG and/or
RNG</td></tr>
</table>

Algorithm Tested		**Additional Required Test(s)**
	MAC algorithm used by the IUT to generate the KDF	CMAC and/or HMAC
KDF (NIST SP 800-135)	IKEv1 KDF	HMAC
	IKEv2 KDF	HMAC
	TLS 1.0/1.1	HMAC SHA-1
	TLS 1.2	HMAC (SHA-256, 384, 512)
	X9.63-2001	SHA
	SSH	SHA
	SRTP	AES using any mode of operation that utilizes the forward cipher function.
	SNMP	SHA-1
	TPM	HMAC 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 (FIPS 186-2 or FIPS 186-3)	KeyGen9.31	RNG or DRBG
		The supported SHA algorithms: SHA-1, SHA-256, SHA-384, or SHA-512 (SHA-224 is allowed in FIPS 186-3 only)
	SigGen9.31	The supported SHA algorithms: SHA-1, SHA-256, SHA-384, or SHA-512 (SHA-224 is allowed in FIPS 186-3 only)
	SigGenPKCS1.5	The supported SHA algorithms: SHA-1, SHA-224, SHA-256, SHA-384, or SHA-512
	SigGenPSS	The supported SHA algorithms: SHA-1, SHA-224, SHA-256, SHA-384, or SHA-512
	SigVer9.31	The supported SHA algorithms: SHA-1, SHA-256, SHA-384, or

Algorithm Tested		Additional Required Test(s)
		SHA-512 (SHA-224 is allowed in FIPS 186-3 only)
	SigVerPKCS1.5	The supported SHA algorithms: SHA-1, SHA-224**,** SHA-256, SHA-384, or SHA-512
	SigGenPSS	The supported SHA algorithms: SHA-1, SHA-224**,** SHA-256, SHA-384, or SHA-512
TDES Counter Mode		TDES using any mode of operation that utilizes the forward cipher function.
XTS AES (NIST SP 800-38E)	Encrypt	AES – any mode of operation that utilizes the forward cipher function.
	Decrypt	AES —any mode of operation that utilizes the forward and inverse cipher function (i.e., AES-ECB or AES-CBC).

GEN.6 An algorithm implementation has restrictions on its use 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 CST laboratory.

GEN.8 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.9 Suppose an algorithm implementation has been validated. What happens when a change is made inside the implementation's boundary? It is claimed that no cryptographic functions were changed. Is the algorithm validation still valid?

No. Any change inside the defined boundary of the implementation creates a new implementation, which must be validated. It does not matter what the change is.

GEN.10 If a vendor claims that their implementation runs on multiple operating systems, how should this be validated?

CAVP validations list the operating system and processor, known as the operating environment (OE), on which the testing was conducted. They do not list all OEs on which the implementation is able to run. Therefore, only one set of tests is required. The OE used for module validation must match the OE used for algorithm validation.

The above applies to a single implementation, such as a single binary executable file or dynamic library. The above does not apply to a vendor who has multiple implementations that use the same name and version number, such as a single source

code base that can be compiled to target different OEs. In this case, each distinct implementation must be tested separately.

A vendor is allowed to test on more OEs than required if it so chooses.

GEN.11 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; nothing 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.12 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. [REF: **FIPS 140-2 IG 1.4**]

For Software implementations, the following information must be listed:

1. Processor – This field shall identify the vendor and processor family. Examples that satisfy the requirement for the processor field are Intel Core i5, ARM 7, and AMD Opteron.

   No further specificity is required **unless** the vendor or the lab knows that the software implementation executes differently on different processors within the same family. In this case, the listing must be more specific. **AES.2** describes such a case. A vendor may also choose to be more specific than required, e.g., in order to gain a marketing benefit.

2. Operating system – This field shall identify the vendor and operating system family, or major version number where more appropriate. Examples that satisfy the requirement for the OS field are Microsoft Windows Vista, Apple Mac OS X, Red Hat Enterprise Linux 5, and Wind River VxWorks 6.

   No further specificity is required **unless** the vendor or lab knows that the software implementation executes differently on different OSes within the

same OS family or major version number. In this case, the listing must be more specific. A vendor may also choose to be more specific than required if so desired.

Any virtual machine (VM) used during testing shall be listed in the OS field of the Operating Environment (OE). If the VM was running between the software implementation and the OS, as in the case of a Java VM, it should be listed along with the OS using the same vendor and family/version number requirements. See **GEN.17** for the case of a VM running underneath the target OS.

For Firmware implementations, the following information must be listed:

1. Processor – This field has the same requirements as the Software Processor field above.

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 **FIPS 140-2 IG G11**, a simulator may be used to test the algorithm implementation. The algorithm implementation would be extracted from the rest of 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. Examples of simulator names include Cadence NC-Verilog, Mentor Graphics ModelSim 10, and Synopsys VCS.

GEN.13 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 an explanation of this and the validation number for the underlying algorithm should list the 140 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 validation number listed will be the 140-1 Module Certificate number.

GEN.14 Can a vendor still get a hard copy algorithm validation certificate?

No. Effective January 1, 2008, the Cryptographic Algorithm Validation Program (CAVP) stopped issuing algorithm validation certificates for cryptographic algorithm validations. The cryptographic algorithm implementation validation entry (found on the appropriate CAVP website validation list) will serve as the official posting of the validation.

GEN.15 Which is the most significant bit and byte in all CAVS vectors?

The most significant bit in all CAVS vectors is the leftmost bit. The most significant byte in all CAVS vectors is the leftmost byte.

Some inputs are bit strings – ordered sequences of bits – instead of numbers. For compactness, CAVS represents each group of 4 (four) bits in a bit string by its hexadecimal number representation, with the left-most bit as the most significant. For example, 10010010 → 1001 0010 → 92 hex.

GEN.16 One implementation tests multiple algorithms. Can one algorithm be tested on one version of CAVS and another algorithm be tested on a different version of CAVS?

Yes, an implementation that tests multiple algorithms may be tested on different versions of CAVS. This would be recorded in the testing files by the CAVS tool. Some examples of this situation include

1. A vender has an implementation of AES and SHA. They originally only get AES validated. At a later date when a later version of CAVS is now active, they request the testing of SHA. Nothing has changed within the algorithmic boundary. SHA will be tested under a different version of CAVS.
2. A vendor has an implementation of AES and SHA. They submit both for testing but have issues with one. So they only get one validated at this time. At a later date the other algorithm is ready for testing and the current version of CAVS is different.

GEN.17 Does CAVP allow algorithm tests to be performed on a target operating system running on top of a virtual machine? Or must the target OS be the machine's native OS?

It depends on whether the virtual machine (VM) is part of the operational environment (OE) listing. For example, suppose an algorithm implementation runs on Windows XP on top of Parallels Desktop for Mac on top of Mac OS X on an Intel Core 2 Duo processor. It is acceptable to list the Operating System (OS) in the OE as "Windows XP on Parallels Desktop for Mac on Mac OS X." It is not acceptable to list the OS as "Windows XP" only. Remember, of course, that the OE listing on the algorithm validation must match the OE in the CMVP submission.

At this point in time we do not feel comfortable treating the OS running on top of a virtual machine as equivalent to the OS running directly on the processor. Probably, for many crypto algorithm implementations, the behavior and results would be the same. However, in discussions with one of our local VM experts, we have identified some differences that might arise. We don't know for certain that these differences would affect validations, but until we run a significant sample of tests, we want to play it safe.

GEN.18 Does GEN.2 make the allowance for a procedurally controlled correct implementation of an algorithm? I.e., can one simply specify in a Security Policy that the calls X, Y, and Z must be called in a particular sequence and manner for an algorithm to maintain its validation?

No. In the "However..." part of GEN.2, it mentions that the order of operations is fixed so there is no way for the component functions to be called out of order. It doesn't get handled in the Security Policy.

GEN.19 In order to apply GEN.9 to a software algorithm, what is considered the implementation boundary of the software algorithm? Is it the source code functions of the algorithm; the source code files that contain those factions; or the binary executable or dynamic library that the source code compiles into?

For a software algorithm, CAVP validates the binary executable or dynamic library that contains the executable code of the algorithm implementation. The implementation's boundary is effectively the entire binary file that contains the algorithm implementation. So to apply GEN.9, if that binary file is different the algorithms in it must be tested again. It does not matter what the change is or that the algorithm source or compilation result may not have changed.

GEN.20 Source code for a cryptographic algorithm is compiled into two separate, non-identical binary files. Can the two binary (executable) files be considered a single implementation? (For example, AES is compiled statically into both an encrypted key storage system and a network encryption system inside the same product; or DSA verification is compiled into a pre-boot loader and into the main program that the pre-boot loader loads.)

No. The CAVP validation for a software or firmware implementation is specific to the binary executable file that the tested implementation resides in. Compiling the same source code into two different executables is considered two separate algorithm implementations and each one needs to be independently tested. It does not matter that the same source code is used.

GEN.21 Can a vendor request that an algorithm implementation be validated but not posted on the validation list until a later date?

No. When a validation number is assigned to the implementation, it must be posted on the CAVP algorithm validation list.

If the vendor does not want this algorithm implementation to be publicly recognized until a later date, the vendor can assign a temporary implementation name when the implementation is submitted to the CAVP by the testing lab. The fact that a temporary name has been assigned to this implementation is transparent to the CAVP; the name of an algorithm implementation is the responsibility of the vendor. The rest of the implementation information displayed on CAVP web site – the vendor information, the versioning, Operational Environment and the Description/Notes field- shall represent the information about what was actually tested,

The vendor may request the lab to submit an official change request to the CAVP if they wish to post the official implementation name. Please see the ***CAVP Algorithm Submission Guidance*** for more information on submitting a change request for an existing validation.

An exception to this guidance is ITAR validations. See the ***CAVP Management Manual*** for instructions on processing ITAR requests.

GEN.22 Suppose you have two implementations that you are testing at the same time and each have their own separate cryptographic algorithm boundaries. Each is a prerequisite of the other. How do you get the validated since neither is validated at the time you are testing the other? For example,

You have the following implementations that have their own separate cryptographic boundaries:

1. ***An implementation of DRBG***
2. ***An implementation of ECDSA***

The prerequisite for DRBG is ECDSA and the prerequisite for ECDSA is DRBG.

If 2 algorithmic implementations have separate cryptographic boundaries and each of these implementations require each other as prerequisites, submit both implementations simultaneously with "Other Special Requests and Notes for CAVP" checked. Include the explanation, in each submission, that you are submitting 2 separate implementations for validation and each is used by the other as prerequisites and therefore the assigned validation number will need to be supplied by the CAVP at processing time.

3 AES FAQ

AES.1 What is required to get an AES Counter Mode implementation validated?

The requirements for getting an AES Counter (CTR) Mode implementation validated are passing the CAVS tests for the underlying forward cipher function used by the CTR mode and validating the counter implementation as described in the AES Validation System (AESAVS) documentation, Appendix A: Counter Mode Requirements.

AES.2 A software implementation of AES uses the AES-NI instruction set. How do I validate it?

There are two separate cases:

1. The implementation relies on the AES-NI instructions and only runs on processors that support them. One set of AES test vectors is needed to validate this implementation. The OE listing for the processor must indicate that the processor supports AES-NI, e.g., "Intel Core i5 with AES-NI w/ Windows 7".

2. The implementation uses AES-NI on processors that support it and does not use AES-NI (i.e., implements AES entirely in software) on processors that do not support it. Thus, there are two distinct execution paths in the code for AES depending upon whether or not the processor supports AES-NI. In this case, two sets of test vectors are needed to validate the implementation: one runs on a processor with AES-NI that uses the AES instructions, and one runs on a processor without AES-NI.

The vendor may choose to have one validation listing with two entries in the OE field or two separate validation listings, each with a different entry in the OE field. An example of the two OE field entries is: "Intel Core i5 with AES-NI w/ Windows 7" and "Intel Core i5 w/ Windows 7."

The above requirements apply to the fundamental (base) modes of operation of AES and to any algorithm that uses AES, whether extended modes of operation such as CCM, GCM, and XTS, or other functions such as the CTR_DRBG and CMAC.

AES.3 What fundamental (or base) modes of operation use the forward cipher function?

ECB in the encrypt state only
CBC in the encrypt state only

CFB in both the encrypt and decrypt states
OFB in both the encrypt and decrypt states

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 CST laboratory prior to February 9, 2005, will be completed. See the DES Transition Plan for more details.

5 Triple-DES FAQ

TDES.1 What is required to get an TDES Counter Mode implementation validated?

The requirements for getting a TDES Counter (CTR) Mode implementation validated are passing the CAVS tests for the underlying forward cipher function used by the CTR mode and validating the counter implementation as described in the AES Validation System (AESAVS) documentation, Appendix A: Counter Mode Requirements. The counter requirements are the same for TDES and AES.

TDES.2 Our TDES implementation does not allow the use of weak keys, but the Known Answer Tests (KATs) in the validation suite use weak keys and, therefore, the implementation needs to be able to accept them. How should we test this?

For validation testing, tighten the algorithmic boundary so that it does not include the weak key check. Make sure the implementation does not allow weak keys outside the validation testing.

TDES.3 Why do the TDES Known Answer Tests (KATs) use weak keys?

The TDES Known Answer Tests (KATs) were based on the DES KATs. Likewise, the DES KATs were based on the standard DES test set described in NIST SP 500-20, "Validating the Correctness of Hardware Implementations of the NBS Data Encryption Standard," written in 1977. These tests were generated before the realization of the weak keys. The purpose of this test is to test every element of the DES (TDES) components. When talking about the key tests, we are referring to the testing of the key permutation tables PC1 and PC2. As detailed in Section 3.1.1.3, "The Variable Key Known Answer Test for the Encryption Process" in NIST SP 800-17:

"When this test is performed for an IUT of the DES algorithm, the 56 possible basis vectors which yield unique keys are presented to PC1 verifying the key permutation, PC1. Since the key schedule consists of left shifts, as *i* ranges over the index set, a complete set of basis vectors is presented to PC2 as well, so this is verified."

TDES.4 What fundamental modes of operation use the forward cipher function?

ECB in the encrypt state only

CBC in the encrypt state only
CBC-I in the encrypt state only
CFB in both the encrypt and decrypt states
CFB-P in both the encrypt and decrypt states
OFB in both the encrypt and decrypt states
OFB-I in both the encrypt and decrypt states

6 DSA FAQ

DSA.1 If a vendor is having problems getting one of the DSA functions to work properly, where can a known set of values be obtained to help in the testing?

Test vectors for all validation tests for every supported algorithm can be found on the CAVP main page in the appropriate algorithm section. This is where sample known sets of values can be found.
If the implementation being tested does not compute the same signature or result, then it can be concluded that there is something wrong with the vendor's 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 What random number generators can implementations of FIPS186-3 DSA use?

Any random number generator (RNG) or random bit generator (RBG) that is **approved** for use in FIPS 140-validated modules may be used, subject to the transition schedule specified in SP 800-131A. Specific references to SP 800-90 should not be considered as a requirement to use a generator specified in SP 800-90 until such time as the use of the other generators is no longer allowed.

7 SHA FAQ

SHA.1 A vendor wants to test SHA (byte only). The vendor's implementation restricts the size of the data string that is hashed, i.e., For SHA-1, data length is less than or equal to 256 bytes. How is this implementation validated?

Currently the CAVS tool assumes there are no restrictions on the size of the data string that can be hashed by the implementation being tested. Therefore, the request file generated by the CAVS tool to test this implementation will contain a wide range of data string lengths. The tester will evaluate only the data string lengths supported by their implementation, returning the answers to the CST laboratory. The CST laboratory will check the response files manually to determine if the data strings supported by the implementation's specifications pass successfully.

In the algorithm validation request cover letter (and email request), the CST Lab will indicate the special case and will explain how the files were verified. The restriction will be indicated on the algorithm's 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 one of the RSA functions to work properly, where can a known set of values be obtained to help in the testing?

Test vectors for all validation tests for every supported algorithm can be found on the CAVP main page in the appropriate algorithm section. This is where sample known sets of values can be found.

If the implementation being tested does not compute the same signature or result, 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?

This question arose before 186-3 RSA was available.

When testing implementations of 186-3 RSA, a different salt length (and salt value, if applicable) can be specified per SHA algorithm/mod size combination supported.

When testing implementations of 186-2 RSA, only one salt length (and salt value, if applicable) can be specified PER SCREEN. This means the same salt length (and salt value, if applicable) will be applied to each combination of SHAs and mod sizes selected on the screen. If different salt lengths (and values) are to be tested for each SHA/mod size combination, the different combinations must be run as separate tests; as separate folders or projects.

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 returned a "Fatal Error" message and indicated that the vectors were generated but with errors. Why did this happen?

PKCS #1, versions 2.1, contains the statement:

"If *emlen* < *hLen* + *sLen* + 2, output "encoding error" and stop",

where *emLen* is ⌈(modulus _length – 1)/8 ⌉, *hlen* is the length of the output block of a hash function (in octets), and *slen* is the length of a salt (in octets). Typical salt lengths in octets are *hLen* and 0.

Section 5.5 of FIPS 186-3, item e, contains the statement;

> "For RSASSA-PSS, the length of the salt (*sLen*) **shall** be: $0 \leq sLen \leq hlen$, where *hlen* is the length of the hash function output block (in bytes or octets)."

These statements are appropriate for the 2048 and 3072-bit moduli for all **approved** hash functions. However, when a 1024-bit modulus is used with SHA-512 and a salt length equal to *hlen* (512 bits = 64 octets, in this case), then:

> *emLen* = ⌈(1024 – 1)/8⌉ = 128 octets,
>
> *hLen* = *sLen* = 64 octets, and
>
> *hLen* + *sLen* + 2 = 130, which is greater than *emLen*, so the process produces an error (see the statement in PKCS #1 that is provided above).

FIPS 186-3 Change Notice modifies item e) of Section 5.5 in FIPS 186-3 accordingly:

> "For RSASSA-PSS:
>
> > If *nlen* = 1024 bits, and the output length of the **approved** hash function output block is 512 bits, then the length of the salt **shall** be $0 \leq sLen \leq hLen - 2$.
> >
> > Otherwise, the length of the salt (*sLen*) **shall** be: $0 \leq sLen \leq hLen$
>
> where *hlen* is the length of the hash function output block (in bytes or octets)."

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

This question is no longer relevant. The error meaning has been added to the fax files.

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 An IUT's RSA Key Generation function 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.

RSA.7 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.

See RSA.5.

RSA.8 From RSA.7 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.

RSA.9 What random number generators can implementations of FIPS186-3 RSA use?

Any random number generator (RNG) or random bit generator (RBG) that is approved for use in FIPS 140-validated modules may be used, subject to the transition schedule specified in SP 800-131A. Specific references to SP 800-90 should not be considered as a requirement to use a generator specified in SP 800-90 until such time as the use of the other generators is no longer allowed.

10 HMAC FAQ

HMAC.1 If an implementation supports other MAC sizes 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?

Yes, when an implementation of the HMAC algorithm is validated, the CAVP requires that the SHA algorithm used by the HMAC implementation be 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 the latest IEEE 802.11 standard, which is 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 listing 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 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.

CCM.4 When testing a CCM implementation, the CAVS screen only allows the associated data length and payload lengths to be between 0 and 32 or 2^16. Why does the CAVP put this restriction on what can be tested?

It is not possible to test every value supported by the IUT for these variables. So instead we allow the minimum value, a mid value, and a maximum value supported by the IUT to be tested. This provides good coverage for testing. It does not imply that an implementation can only implement these lengths.

If the vendor would like to indicate all the actual lengths for associated data and payload that are supported by the IUT, this information can be listed in the description.

12 ECDSA FAQ

ECDSA.1 For ECDSA PKV validation testing, how are the Qx and Qy values represented? What is the significance of their representation?

All values should be thought of as hexadecimal numbers. To determine whether a value or a point is valid, convert it to a number; do not rely on the format (e.g., leading zeros, number of hexadecimal symbols, etc.)

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

This question is no longer relevant. The error meaning has been added to the fax files.

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 validation.
2. In order to receive an ECDSA algorithm (FIPS 186-2 or FIPS 186-3) validation, 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 validation.
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.

ECDSA.4 Can an ECDSA implementation that uses SHA2 be tested?

Yes. FIPS 186-3 ECDSA implementations may use the SHA-2 (i.e., SHA-224, SHA-256, SHA-384, or SHA-512) algorithms. Any ECDSA implementation that uses one of the SHA2 algorithms must be validated for conformance to FIPS 186-3. An implementation that uses SHA-1 may be validated for conformance to either FIPS 186-2 or FIPS 186-3 ECDSA.

ECDSA.5 What random number generators can implementations of FIPS186-3 ECDSA use?

Any random number generator (RNG) or random bit generator (RBG) that is approved for use in FIPS 140-validated modules may be used, subject to the transition schedule specified in SP 800-131A. Specific references to SP 800-90 should not be considered as a requirement to use a generator specified in SP 800-90 until such time as the use of the other generators is no longer allowed.

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.

CMAC.2 Can an implementation that only supports message lengths divisible by the block size be tested? How about implementations that only support partial block sizes? How do I indicate this in the testing? How is it recorded on the AES and TDES Validation listing?

Yes, an implementation can support full block sizes only, partial block sizes only, or both full and partial block sizes. To test an implementation that only supports full block sizes, only provide 'message lengths divisible by the block size' leaving the 'not divisible by block size' section blank (or zero). To test an implementation that only supports partial block sizes, only provide 'message sizes not divisible by block size' size' leaving the 'divisible by block size' section blank (or zero). Both sections are filled in if the implementation supports both full and partial block sizes. On the AES and TDES Validation List website, a CMAC entry will indicate BlockSize: Full, BlockSize: Partial, or BlockSize: Full/Partial.

14 KAS FAQ

KAS.1 For KAS implementations with no key confirmation, is the vendor expected to actually implement and separately test a supporting MAC (e.g., HMAC-SHA-512 for KAS ECC Ephemeral Unified parameter set EE), or would default MAC values of some sort be used only for the purposes of KAS validation testing?

No, for KAS implementations with no key confirmation, the vendor needs a MAC implementation but it doesn't have to be part of the KAS implementation. A MAC implementation is just needed for purposes of KAS validation testing.

KAS.2 If no MAC is actually required to be implemented in cases where key confirmation is not supported, and the MAC is only being specified in the tool for the purposes of KAS testing, which MAC should be selected for parameter sets with multiple options? Are there default values that should be entered for whichever MAC is selected since this is a mandatory field?

No, It doesn't matter. If you have parameter sets with multiple options, this will be reflected in the shared secret value z and the derived keying material. The CAVS tool just needs a MAC to use the key to see if it gets the correct answer.
For example, if ECC Ephemeral Unified NOKC with parameter set EC supported both P-521 and B-571 curves and SHAs 256 and 512, you can select only one MAC to be used for testing all of these options to assure that the derived keying material is correct.

KAS.3 Where can definitions be found of the variable names in the testing files?

All the variables in the KAS testing files are defined in the KASVS document.

15 GCM FAQ

GCM.1 What do I need to do to validate an AES-GCM (NIST SP 800-38D) algorithm implementation?

All GCM implementations must pass the CAVS tests. GCM implementations that use an externally-generated initialization vector (IV) have no other requirements. GCM implementations that use an internally-generated IV have one additional requirement: the CST lab must verify that the IV is generated using either the method of Section 8.2.1 or the method of Section 8.2.2 of NIST SP 800-38D.

GCM.2 How should I verify that the IV is generated using the method of Section 8.2.1 or Section 8.2.2?

The lab can use any technique that it deems suitable. The CAVP leaves the decision up to the CST laboratory.

GCM.3 Why does CAVS allow testing for an externally generated IV when Section 9.1 of NIST SP 800-38D says the module must generate IVs within the module boundary? (FIPS 140-2 IG 1.14 has a similar statement).

The CAVP validates cryptographic algorithm implementations, not cryptographic modules. The cryptographic algorithm boundary does not have to be the same as the cryptographic module boundary. In many cases the cryptographic algorithm implementation is itself a module, but in other cases it is part of (i.e., a component of) a cryptographic module. Thus, for example, a validated AES GCM implementation could be combined with a validated Approved RBG (e.g., a NIST SP 800-90 DRBG or a FIPS 186-2 RNG) implementation that will generate IVs in a crypto module. The requirements in Section 9.1 of NIST SP 800-38D apply to a cryptographic module and module validation under FIPS 140-2, as does the similar text in FIPS 140-2 IG 1.14.

GCM.4 Are there any prerequisites for validating a GCM implementation?

Please refer to GEN.5 where the prerequisites for all algorithms are listed.

16 DRBG FAQ

DRBG.1 Is ANSI X9.62-2005 DRBG the same as NIST SP 800-90A HMAC_DRBG?

Yes, they are the same. NIST SP 800-90A has a nonce as one of the inputs to the instantiate function, which is not in ANSI X9.62-2005. However, NIST SP 800-90A Section 8.6.7 specifies that additional entropy input can be used in place of a nonce, making it the same as ANSI X9.62-2005.

DRBG.2 How does an implementation of ANSI X9.62-2005 DRBG get validated?

It gets validated by passing the NIST SP 800-90A HMAC_DRGB tests. It would be issued an NIST SP 800-90A validation and be listed as HMAC_DRBG.

DRBG.3 Does CTR_DRBG use AES (or TDES) in counter mode or, as NIST SP 800-90A Sections 10.2.1 and 10.4.3 seem to indicate, in ECB mode?

In AES counter mode (or TDES counter mode), the forward cipher function (sometimes called the encrypt function) has a counter value as the input instead of plaintext (ECB mode), a chained ciphertext value (CBC mode), or some other value. The counter has an initial value, not necessarily 0, and is incremented between calls to the forward cipher function. NIST SP 800-90A Section 10.2.1.5.1 Step 4 implements the counter mode. `V` is the counter; it is incremented at the beginning of each pass through the loop. The `Block_Encrypt` (forward cipher function) operation is performed on the counter and the result is appended to the end of the `temp` bitstring.

The only difference between how counter mode is used here and how it is used for encryption is that when used for encryption, the result of the "encrypt counter" operation is XOR'ed with the plaintext. Here, inside the DRBG, we are not encrypting anything, so result is returned as pseudorandom bits and there is no XOR step. The Update function in NIST SP 800-90A 10.2.1.2 (`CTR_DRBG_Update`), Step 2 implements a counter in the same way and does include an XOR of the result with `provided_data` to produce an updated internal state (`Key` and `V`).

DRBG.4 We validated our DRBG implementation before NIST SP 800-90A came out. Do we have to do a new validation?

No. NIST SP 800-90A is a revision of NIST SP 800-90. The DRBG mechanisms have not changed. The CAVS tests have not changed. SP 800-90A does allow use of the two new SHA functions from FIPS 180-4 (SHA-512/224 and SHA-512/256). When CAVS SHA and HMAC testing is available for the new SHA functions, they will be added as options for DRBG testing as well.

DRBG implementations validated against NIST SP 800-90 are considered validated against NIST SP 800-90A as well. There is no change in the testing. The same validation list (i.e., DRBG Validation List) is used, and no distinction is made between implementations validated before and after NIST SP 800-90A was published.

There was one minor change in the CAVS DRBG tests in the first release following the publication of NIST SP 800-90A. The number of returned bits in the call to generate was changed from a fixed value, equal to one output block, to a test parameter ranging from 1 to 32 (default 4) output blocks. This was not a result of any new requirements in NIST SP 800-90A.