Apple Inc.

Apple iOS CoreCrypto Module, v5.0
FIPS 140-2 Non-Proprietary Security Policy

Document Control Number
FIPS_CORECRYPTO_IOS_US_SECPOL_1.3
Version 1.3

June, 2015

Prepared for:
Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
www.apple.com

Prepared by:
atsec information security Corp.
9130 Jollyville Road, Suite 260
Austin, TX 78759
www.atsec.com

Table of Contents

List of Tables

List of Figures

1 Introduction

1.1 Purpose

This document is a non-proprietary Security Policy for the Apple iOS CoreCrypto Module, v5.0. It describes the module and the FIPS 140-2 cryptographic services it provides. This document also defines the FIPS 140-2 security rules for operating the module.

This document was prepared in fulfillment of the FIPS 140-2 requirements for cryptographic modules and is intended for security officers, developers, system administrators, and end-users.

FIPS 140-2 details the requirements of the Governments of the U.S. and Canada for cryptographic modules, aimed at the objective of protecting sensitive but unclassified information.

For more information on the FIPS 140-2 standard and validation program please refer to the NIST website at http://csrc.nist.gov/cryptval.

Throughout the document "Apple iOS CoreCrypto Module, v5.0." "cryptographic module", "CoreCrypto" or "the module" are used interchangeably to refer to the Apple iOS CoreCrypto Module, v5.0.

1.2 Document Organization / Copyright

This non-proprietary Security Policy document may be reproduced and distributed only in its original entirety without any revision, ©2015 Apple Inc.

1.3 External Resources / References

The Apple website (http://www.apple.com) contains information on the full line of products from Apple Inc. For a detailed overview of the operating system iOS and its security properties refer to [iOS] and [SEC]. For details on iOS releases with their corresponding validated modules and Crypto Officer Role Guides refer to the Apple Knowledge Base Article HT202739 - "iOS product security: Validations and guidance" (https://support.apple.com/en-us/HT202739)

The Cryptographic Module Validation Program website (http://csrc.nist.gov/groups/STM/cmvp/index.html) contains links to the FIPS 140-2 certificate and Apple, Inc. contact information.

1.3.1 Additional References

FIPS 140-2	Federal Information Processing Standards Publication, "FIPS PUB 140-2 Security Requirements for Cryptographic Modules," Issued May-25-2001, Effective 15-Nov-2001, Location: http://csrc.nist.gov/groups/STM/cmvp/standards.html
FIPS 180-4	Federal Information Processing Standards Publication 180-4, March 2012, Secure Hash Standard (SHS)
FIPS 197	Federal Information Processing Standards Publication 197, November 26, 2001 Announcing the ADVANCED ENCRYPTION STANDARD (AES)
PKCS7	RSA Laboratories, "PKCS#7 v1.5: Cryptographic Message Syntax Standard," 1993. Location: http://www.rsa.com/rsalabs/node.asp?id=2129
PKCS3	RSA Laboratories, "PKCS#3 v1.4: Diffie-Hellman Key Agreement Standard," 1993. Location: http://www.rsa.com/rsalabs/node.asp?id=2126
IG	NIST, "Implementation Guidance for FIPS PUB 140-2 and the Cryptographic Module Validation Program," December 21, 2012 Location: http://csrc.nist.gov/groups/STM/cmvp/standards.html

iOS	iOS Technical Overview Location: http://developer.apple.com/library/ios/#documentation/Miscellaneous/ Conceptual/iPhoneOSTechOverview/Introduction/Introduction.html#//apple_ref/doc/uid/TP40007898
SEC	Security Overview Location: http://developer.apple.com/library/ios/#documentation/Security/ Conceptual/Security_Overview/Introduction/Introduction.html
SP800-57P1	NIST Special Publication 800-57, “Recommendation for Key Management – Part 1: General (Revised),” July 2012
SP 800-90A	NIST Special Publication 800-90A, “Recommendation for Random Number Generation Using Deterministic Random Bit Generators,” January 2012
UG	User Guide Location: http://developer.apple.com/library/ios/navigation/

1.4 Acronyms

Acronyms found in this document are defined as follows:

AES	Advanced Encryption Standard
BS	Block Size
CAVP	Cryptographic Algorithm Validation Program
CBC	Cipher Block Chaining mode of operation
CFB	Cipher Feedback mode of operation
CMVP	Cryptographic Module Validation Program
CSP	Critical Security Parameter
CTR	Counter mode of operation
DES	Data Encryption Standard
DH	Diffie-Hellmann
DMA	Direct Memory Access
DRBG	Deterministic Random Bit Generator
DS	Digest Size
ECB	Electronic Codebook mode of operation
ECC	Elliptic Curve Cryptography
EC Diffie-Hellman	DH based on ECC
ECDSA	DSA based on ECC
E/D	Encrypt/Decrypt
EMC	Electromagnetic Compatibility
EMI	Electromagnetic Interference
FIPS	Federal Information Processing Standard

FIPS PUB	FIPS Publication
GCM	Galois/Counter Mode
HMAC	Hash-Based Message Authentication Code
HW	Hardware
KAT	Known Answer Test
KEK	Key Encryption Key
KEXT	Kernel extension
KDF	Key Derivation Function
KO 1	Triple-DES Keying Option 1: All three keys are independent
API	Kernel Programming Interface
KS	Key Size (Length)
MAC	Message Authentication Code
NIST	National Institute of Standards and Technology
OFB	Output Feedback (mode of operation)
OS	Operating System
PBKDF	Password-based Key Derivation Function
PWCT	Pair Wise Consistency Test
RNG	Random Number Generator
SHS	Secure Hash Standard
SW	Software
Triple-DES	Triple Data Encryption Standard
TLS	Transport Layer Security

2 Cryptographic Module Specification

2.1 Module Description

The Apple iOS CoreCrypto Module, v5.0 is a software cryptographic module running on a multi-chip standalone mobile device.

The cryptographic services provided by the module are:

- Data encryption / decryption
- Generation of hash values
- Key wrapping
- Message authentication
- Random number generation
- Key generation
- Signature generation / verification
- Key derivation

2.1.1 Module Validation Level

The module is intended to meet requirements of FIPS 140-2 security level 1 overall. The following table shows the security level for each of the eleven requirement areas of the validation.

FIPS 140-2 Security Requirement Area	Security Level
Cryptographic Module Specification	1
Cryptographic Module Ports and Interfaces	1
Roles, Services and Authentication	1
Finite State Model	1
Physical Security	N/A
Operational Environment	1
Cryptographic Key Management	1
EMI/EMC	1
Self-Tests	1
Design Assurance	1
Mitigation of Other Attacks	1

Table 1: Module Validation Level

2.1.2 Module Components

In the following sections the components of the Apple iOS CoreCrypto Module, v5.0 are listed in detail. There are no components excluded from the validation testing.

2.1.2.1 Software components

CoreCrypto has an API layer that provides consistent interfaces to the supported algorithms. These implementations include proprietary optimizations of algorithms that are fitted into the CoreCrypto framework.

2.1.2.2 Hardware components

The AES hardware accelerator, if present, is integrated into the CPU of the system as referenced in Table 2.

2.1.3 Tested Platforms

The module has been tested on the following platforms:

Manufacturer	Model	Operating System
Apple Inc.	iPhone4S with Apple A5 CPU (with and without AES hardware accelerator)	iOS 8.0
Apple Inc.	iPhone5 with Apple A6 CPU (with and without AES hardware accelerator)	iOS 8.0
Apple Inc.	iPhone5S with Apple A7 CPU	iOS 8.0
Apple Inc.	iPad (3rd Generation) with Apple A5X CPU (with and without AES hardware accelerator)	iOS 8.0
Apple Inc.	iPad (4th Generation) with Apple A6X CPU (with and without AES hardware accelerator)	iOS 8.0
Apple Inc.	iPhone6 with Apple A8 CPU (iPhone6 and iPhone6 Plus)	iOS 8.0
Apple Inc.	iPad Air 2 with Apple A8X CPU	iOS 8.0

Marketing name for iPad (3rd generation) is 'New iPad'.

Table 2: Tested Platforms

2.2 Modes of Operation

The Apple iOS CoreCrypto Module, v5.0 has an Approved and non-Approved modes of operation. The Approved mode of operation is configured by default and cannot be changed. If the device starts up successfully then CoreCrypto framework has passed all self-tests and is operating in the Approved mode. Any calls to the non-Approved security functions listed in Table 4 will cause the module to assume the non-Approved mode of operation.

The module transitions back into FIPS mode immediately when invoking one of the approved ciphers as all keys and Critical Security Parameters (CSP) handled by the module are ephemeral and there are no keys and CSPs shared between any functions. A re-invocation of the self-tests or integrity tests is not required.

Even when using this FIPS 140-2 non-approved mode, the module configuration ensures that the self-tests are always performed during initialization time of the module.

The module contains multiple implementations of the same cipher as listed below. If multiple implementations of the same cipher are present, the module selects automatically which cipher is used based on internal heuristics. This includes the hardware-assisted AES (AES support offered by the CPU) implementation where available.

The Approved security functions are listed in Table 3. Column four (Val. No.) lists the validation numbers obtained from NIST for successful validation testing of the implementation of the cryptographic algorithms on the platforms as shown in Table 2 under CAVP.

Refer to http://csrc.nist.gov/groups/STM/cavp/index.html for the current standards, test requirements, and special abbreviations used in the following table.

Approved Security Functions

Cryptographic Function	Standards	Usage / Description	Validation Number						
			A5 CPU	A5X CPU	A6 CPU	A6X CPU	A7 CPU	A8 CPU	A8X CPU
Triple-DES	ANSIX9.52-1 998 FIPS 46-3 SP 800-67 SP 800-38A Appendix E	Encryption / decryption with all keys independent Block chaining modes: ECB, CBC, CFB8, CFB64, OFB, CTR with internal counter	#1856	#1857	#1858	#1859	#1860 #1862	#1861 #1863	#1910 #1920
AES	FIPS 197 SP 800-38 A SP 800-38 D	Generic-software implementation (non-optimized based on LibTomCrypt): Encryption / decryption Key sizes: 128 bits, 192 bits, 256 bits Block chaining modes: ECB, CBC, CFB8, CFB128, OFB, CTR with internal counter, GCM with tag lengths of 128, 120, 112, 104, 96, 64, 32	#3267	#3268	#3269	#3270	#3271 #3273	#3272 #3274	#3379 #3381
		Generic-software implementation (non-optimized based on Gladman): Encryption / decryption Key sizes: 128 bits, 192 bits, 256 bits Block chaining modes: CBC	#3021	#3022	#3023	#3039	#3024 #3025	#3074 #3075	#3376 #3378

Cryptographic Function	Standards	Usage / Description	Validation Number						
			A5 CPU	A5X CPU	A6 CPU	A6X CPU	A7 CPU	A8 CPU	A8X CPU
		Optimized-assembler implementation: Encryption / decryption Key sizes: 128 bits, 192 bits, 256 bits Block chaining modes: ECB, CBC, CFB8, CFB128, OFB, CTR with internal counter, GCM with tag lengths of 128, 120, 112, 104, 96, 64, 32	#3036	#3037	#3015	#3038	#3016 #3017	#3034 #3035	#3355 #3377
		Hardware implementation: AES-CBC Encryption / Decryption Key sizes: 128, 192, 256)	#3018	#3019	#3020	#3040	N/A	N/A	N/A
RSA	FIPS186-4 PKCS #1.5	Key(Gen) sizes(modulus): 2048 bits, 3072 bits SIG(gen) sizes(modulus): 2048 bits, 3072 bits Hash algorithms: SHA-224, SHA-256, SHA-384, SHA-512 SIG(ver) sizes (modulus): 1024 bits, 2048 bits, 3072 bits Hash algorithms: SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	#1667	#1668	#1669	#1670	#1671 #1673	#1672 #1674	#1734 #1736

Cryptographic Function	Standards	Usage / Description	Validation Number						
			A5 CPU	A5X CPU	A6 CPU	A6X CPU	A7 CPU	A8 CPU	A8X CPU
ECDSA	FIPS 186-4 ANSI X9.62	PKG: curves P-256, P-384 PKV: curves P-256, P-384 SIG(gen): curves P-256, P-384 SIG(ver): curves P-256, P-384	#623	#624	#625	#626	#627 #629	#628 #630	#670 #672
SHS	FIPS 180-4	Generic-software implementation (non-optimized): SHA-1 (BYTE-only) SHA-224 (BYTE-only) SHA-256 (BYTE-only) SHA-384 (BYTE-only) SHA-512 (BYTE-only)	#2705	#2706	#2707	#2708	#2709 #2711	#2710 #2712	#2781 #2797
		Optimized-software implementation using VNG: SHA-1 (BYTE-only) SHA-224 (BYTE-only) SHA-256 (BYTE-only)	#2523	#2524	#2525	#2532	#2526 #2527	#2533 #2534	#2796 #2799
HMAC	FIPS 198	Generic-software implementation (non-optimized): KS<BS, KS=BS, KS>BS HMAC-SHA-1 HMAC-SHA-224 HMAC-SHA-256 HMAC-SHA-384 HMAC-SHA-512 Key Size: at least 112 bits	#2066	#2067	#2068	#2069	#2070 #2072	#2071 #2073	#2137 #2152

Cryptographic Function	Standards	Usage / Description	Validation Number						
			A5 CPU	A5X CPU	A6 CPU	A6X CPU	A7 CPU	A8 CPU	A8X CPU
		Optimized-software implementation: KS<BS, KS=BS, KS>BS HMAC-SHA-1 HMAC-SHA-224 HMAC-SHA-256 Key Size: at least 112 bits	#1907	#1908	#1909	#1916	#1910 #1911	#1917 #1918	#2151 #2154
CTR DRBG	SP 800-90A	Generic-software implementation of AES (non-optimized) Key Size: 128 bit	#726	#727	#728	#729	#730 #732	#731 #733	#802 #804
		Optimized-software implementation: AES with 128 bit key size	#583	#584	#575	#585	#576 #577	#581 #582	#800 #801
PBKDF	SP 800-132	Password based key derivation using HMAC with SHA-1 or SHA-2 as pseudorandom function	Vendor Affirmed						

Table 3: Approved Security Functions

CAVEAT: The module generates cryptographic keys whose strengths are modified by available entropy – 160-bits.

Non-Approved Security Functions:

Cryptographic Function	Usage / Description	Caveat
AES	Generic-software implementation of AES (non-optimized based on LibTomCrypt): Encryption / Decryption Block Chaining Mode: CCM	Non-compliant
	Optimized- assembler implementation: Encryption / Decryption Block Chaining Mode: GCM, CTR	Non-compliant
	Optimized- assembler implementation: Encryption / Decryption Generic-software implementation of Block Chaining Mode: CBC	Non-compliant

Cryptographic Function	Usage / Description	Caveat
RSA (encrypt, decrypt) FIPS 186-2	Key wrapping RSAES-OAEP, RSAES-PKCS1-v1_5 PKCS#1 v2.1	Non-Approved, but allowed: RSA (key wrapping; key establishment methodology provides between 112 and 150 bits of encryption strength; non-compliant less than 112 bits of encryption strength)

RSA (sign, verify)	ANSI X9.31
	SIG(gen)
	SIG(ver)
	Hash algorithms: SHA-1, SHA-224, SHA-256, SHA-384, SHA-512
		Non-compliant

	PKCS1-v1_5
	SIG(gen)
	SIG(ver)
	Key sizes (modulus):
	1024-4096 bits in
	multiple of 32 bits not listed in table 3
		Non-compliant

RSA (key pair generation)	ANSI X9.31 Key sizes (modulus): 1024-4096 bits in multiple of 32 bits not listed in table 3 Public key exponent values: 65537 or larger	Non-compliant
ECDSA	Key pair generation for compact point representation of points	Non-Approved
ECDSA	PKG: curves P-192, P-224, P-521 PKV: curves P-192, P-224, P-521 SIG(gen): curves P-192,P-224, P-521 SIG(ver): curves P-192,P-224 P-521	Non-compliant
Integrated Encryption Scheme on elliptic curves	Encryption/Decryption	Non-Approved
Diffie-Hellman	ANSI X9.42, SP 800-56A Key agreement Key sizes: Min 1024 bits, Max 4096 bits	Non-Approved, but allowed: Diffie-Hellman (key agreement; key establishment methodology provides between 112 and 150 bits of encryption strength; non-compliant less than 112 bits of encryption strength).
EC Diffie-Hellman	Key agreement ANSI X9.63, SP 800-56A bit length of ECC subgroup order P-256, P-384	Non-Approved, but allowed: EC Diffie-Hellman (key agreement; key establishment methodology provides 128 bits of encryption strength for P-256 and 160 bits for P-384 - the strength for P-384 is limited by the entropy of the seed source as specified in the caveat).

Ed25519	Key agreement
	Sig(gen)
	Sig(ver)
		Non-Approved

Cryptographic Function	Usage / Description	Caveat
SP800-38F with AES Core	Key Wrapping	Non-Approved
SP800-108	KDF Modes: Counter and Feedback	Non-Approved
SP800-56C	KDF	Non-Approved
ANSI X9.63 KDF	Hash based KDF based on ANSI X9.63	Non-Approved
RFC6637 KDF	KDF based on RFC 6637	Non-Approved
DES	Encryption and decryption: key size 56 bits	Non-Approved
TDES	Optimized-assembler implementation: Encryption / Decryption Block Chaining Mode: CTR	Non-compliant
TDES	Encryption and decryption One-Key and Two-Key implementations	Non-compliant
CAST5	Encryption and decryption: key sizes 40 to 128 bits in 8-bit increments	Non-Approved
RC4	Encryption and decryption: key size 8 to 4096 bits	Non-Approved
RC2	Encryption and decryption: key size 8 to 1024 bits	Non-Approved
MD2	Hashing Digest size 128 bit	Non-Approved
MD4	Hashing Digest size 128 bit	Non-Approved
MD5	Hashing Digest size 128 bit	Non-Approved, but allowed: Used as part of the TLS key establishment scheme only
RIPEMD	Hashing Digest size 128, 160, 256, 320 bits	Non-Approved
Blowfish	Encryption and decryption	Non-Approved
CMAC AES128	MAC generation	Non-Approved
OMAC (One-Key CBC MAC)	MAC generation	Non-Approved
HMAC-DRBG	HMAC based RNG	Non-compliant
Hash-DRBG	Hash-DRBG using SHA1 derived from the SP800-90A without derivation function, without prediction resistance	Non-compliant

Table 4: Non-Approved Functions

The encryption strengths included in Table 4 for the key establishment methods are determined in accordance with FIPS 140-2 Implementation Guidance [IG] section 7.5 and NIST Special Publication 800-57 (Part1) [SP800-57P1].

2.3 Cryptographic Module Boundary

The physical boundary of the module is the physical boundary of the iOS device (iPhone or iPad) that contains the module. Consequently, the embodiment of the module is a multi-chip standalone cryptographic module.

The logical module boundary is depicted in the logical block diagram given in Figure 1.

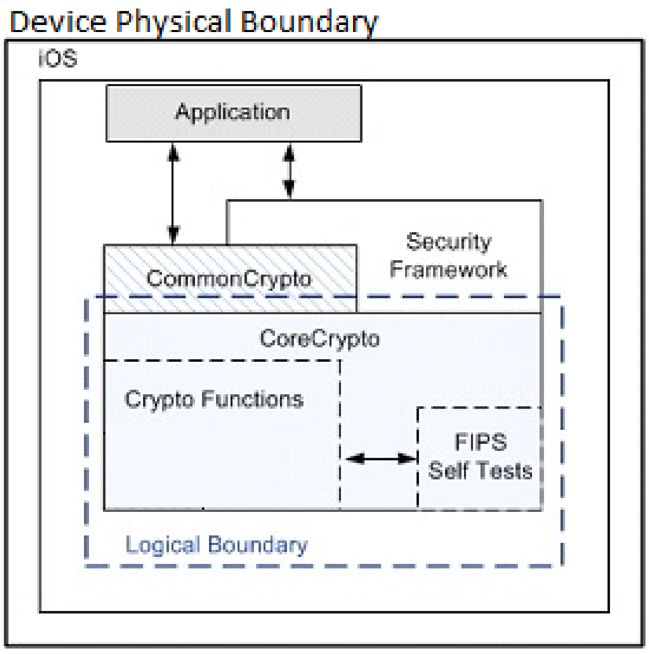

Figure 1: Logical Block Diagram

2.4 Module Usage Considerations

A user of the module must consider the following requirements and restrictions when using the module:

- When using AES-GCM, the caller must use the module's DRBG to generate at least 96 bits of random data that is used for the IV of AES-GCM. The caller is permitted to add additional deterministic data to that IV value in accordance with SP800-38D section 8.2.2. Users should consult SP 800-38D, especially section 8, for all of the details and requirements of using AES-GCM mode.
- When using AES, the caller must obtain a reference to the cipher implementation via the functions of ccaes_[cbc|ecb|…]_[encrypt|decrypt]_mode.
- When using SHA, must obtain a reference to the cipher implementation via the functions ccsha[1|224|256|384|512]_di.
- In case the module's power is lost and then restored, the key used for the AES GCM encryption/decryption **shall** be re-distributed

3 Cryptographic Module Ports and Interfaces

The underlying logical interfaces of the module are the C language Application Programming Interfaces (APIs). In detail these interfaces are the following:

- Data input and data output are provided in the variables passed in the API and callable service invocations, generally through caller-supplied buffers. Hereafter, APIs and callable services will be referred to as “API”.
- Control inputs which control the mode of the module are provided through dedicated parameters and /var/db/FIPS/fips_data holding the HMAC check file
- Status output is provided in return codes and through messages. Documentation for each API lists possible return codes. A complete list of all return codes returned by the C language APIs within the module is provided in the header files and the API documentation. Messages are documented also in the API documentation.

The module is optimized for library use within the iOS user space and does not contain any terminating assertions or exceptions. It is implemented as an iOS dynamically loadable library. The dynamically loadable library is loaded into the iOS application and its cryptographic functions are made available. Any internal error detected by the module is reflected back to the caller with an appropriate return code. The calling iOS application must examine the return code and act accordingly. There are two notable exceptions: (i) ECDSA and RSA do not return a key if the pair-wise consistency test fails; (ii) the DRBG algorithm loops a few iterations internally if the continuous test fails, eventually recovering from the error or causing a shutdown if the problem persists.

The function executing FIPS 140-2 module self-tests does not return an error code but causes the system to crash if any self-test fails – see Section 9.

The module communicates any error status synchronously through the use of its documented return codes, thus indicating the module’s status. It is the responsibility of the caller to handle exceptional conditions in a FIPS 140-2 appropriate manner.

Caller-induced or internal errors do not reveal any sensitive material to callers.

Cryptographic bypass capability is not supported by the module.

4 Roles, Services and Authentication

This section defines the roles, services and authentication mechanisms and methods with respect to the applicable FIPS 140-2 requirements.

4.1 Roles

The module supports a single instance of the two authorized roles: the Crypto Officer and the User. No support is provided for multiple concurrent operators or a Maintenance operator.

Role	**General Responsibilities and Services (details see below)**
User	Utilization of services of the module.
Crypto Officer (CO)	Utilization of services of the module.

Table 5: Roles

4.2 Services

The module provides services to authorized operators of either the User or Crypto Officer roles according to the applicable FIPS 140-2 security requirements.

Table 6 contains the cryptographic functions employed by the module in the Approved and non-Approved mode. For each available service it lists, the associated role, the Critical Security Parameters (CSPs) and cryptographic keys involved, and the type(s) of access to the CSPs and cryptographic keys.

CSPs contain security-related information (for example, secret and private cryptographic keys) whose disclosure or modification can compromise the main security objective of the module, namely the protection of sensitive information.

The access types are denoted as follows:

- 'R': the item is read or referenced by the service
- 'W': the item is written or updated by the service
- 'Z': the persistent item is zeroized by the service

Service	**Roles**		**CSPs & crypto keys**	**Access Type**
	USER	CO		
Triple-DES encryption and decryption Encryption *Input:* plaintext, IV, key *Output:* ciphertext Decryption *Input:* ciphertext, IV, key *Output:* plaintext	X	X	secret key	R

Service	Roles		CSPs & crypto keys	Access Type
	USER	CO		
AES encryption and decryption Encryption *Input:* plaintext, IV, key *Output:* ciphertext Decryption *Input:* ciphertext, IV, key *Output:* plaintext	X	X	secret key	R
Secure Hash Generation *Input:* message *Output:* message digest	X	X	none	N/A
HMAC generation *Input:* HMAC key, message *Output:* HMAC value of message	X	X	secret HMAC key	R
RSA signature generation and verification Signature generation *Input:* the module n, the private key d, the SHA algorithm (SHA -224/SHA-256/SHA-384/SHA-512), a message m to be signed Output: the signature s of the message Signature verification Input: the module n, the public key e, the SHA algorithm (SHA-1/SHA -224/SHA-256/SHA-384/SHA-512), a message m, a signature for the message Output: pass if the signature is valid, fail if the signature is invalid	X	X	secret key	R W

Service	Roles		CSPs & crypto keys	Access Type
	USER	CO		
ECDSA signature generation and verification Signature generation *Input:* message m, q, a, b, X_G, Y_G, n, the SHA algorithm (SHA -224/SHA-256/SHA-384/SHA-512), sender's private key d *Output:* signature of m as a pair of r and s Signature verification *Input:* received message m', signature in form on r' and s' pair, q, a, b, X_G, Y_G, n, sender's public key Q, the SHA algorithm (SHA-1/SHA -224/SHA-256/SHA-384/SHA-512) *Output:* pass if the signature is valid, fail if the signature is invalid	X	X	secret key	R W
Random number generation *Input:* Entropy Input, Nonce, Personalization String *Output:* Returned Bits	X	X	Entropy input string, Nonce, V and K	R W Z
PBKDF Password-based key derivation *Input:* encrypted key and password *Output:* plaintext key or *Input:* plaintext key and password *Output:* encrypted data	X	X	Secret key, password	R W Z
AES key import Input: key Output: N/A	X	X	secret key	R
Triple-DES key import Input: key Output: N/A	X	X	secret key	R
HMAC key import Input: key Output: N/A	X	X	HMAC key	R

Service	**Roles**		**CSPs & crypto keys**	**Access Type**
	USER	CO		
RSA (key pair generation) *Input:* modulus size, the public key, random numbers: X_{p1}, X_{p2}, X_{q1} and X_{q2} *Output:* the private prime factor p, the private prime factor q, the value of the modulus n, the value of the private signature, exponent d	X	X	Asymmetric key pair	R W
Diffie-Hellman Key agreement *Input:* prime number (p), base (g), secret integers(a,b) *Output:* shared secret	X	X	Asymmetric keys (RSA/ECDSA key) and secret session key (AES/Triple-DES key)	R W
EC Diffie-Hellman Key agreement *Input:* domain parameter (p,a,b,G,n,h), key pair (d, Q) *Output:* shared secret	X	X	Asymmetric keys (RSA/ECDSA key) and secret session key (AES/Triple-DES key)	R W
Release all resources of symmetric crypto function context *Input:* context Output: N/A	X	X	AES/Triple-DES key	Z
Release all resources of hash context *Input:* context Output: N/A	X	X	HMAC key	Z
Release of all resources of Diffie-Hellman context for Diffie-Hellman and EC Diffie-Hellman *Input:* context Output: N/A	X	X	Asymmetric keys (RSA/ECDSA key) and secret session key (AES/Triple-DES key)	Z
Release of all resources of asymmetric crypto function context *Input:* context Output: N/A	X	X	RSA/ECDSA keys	Z
Reboot	X	X	N/A	N/A
Self-test	X	X	Software integrity key	R
Show Status	X	X	None	N/A

Table 6: Approved and Allowed Services in Approved Mode

Service	Roles		Access Type
	USER	CO	
AES encryption and decryption Modes: CCM, GCM, CTR, CBC	X	X	R
Integrated Encryption Scheme on elliptic curves encryption and decryption	X	X	R
DES encryption and decryption	X	X	R
TDES encryption and decryption Mode: CTR	X	X	R
TDES encryption and decryption with One-Key and Two-Key implementations	X	X	R
CAST5 encryption and decryption	X	X	R
Blowfish encryption and decryption	X	X	R
RC4 Hash	X	X	R W
RC2 Hash	X	X	R W
MD2 Hash	X	X	R W
MD4 Hash	X	X	R W
MD5 Hash	X	X	R W
RIPEMD Hash	X	X	R W
SP800-38F Key Wrapping with AES Core	X	X	R
RSA Key Wrapping with RSAES-OAEP, RSAES-PKCS1-v1_5 PKCS#1 v2.1	X	X	R
RSA ANSI X9.31 Signature Generation and Verification	X	X	R W

Service	**Roles**		**Access Type**
	USER	CO	
RSA PKCS1-v1_5 Signature Generation and Verification Key sizes: 1024-4096 bits in multiple of 32 bits not listed in table 3	X	X	R W
RSA ANSI X9.31 Key Pair Generation Key sizes (modulus): 1024-4096 bits in multiple of 32 bits not listed in table 3 Public key exponent values: 65537 or larger	X	X	R W
ECDSA Key Pair Generation for compact point representation of points	X	X	R W
ECDSA PKG: curves P-192, P-224, P-521 PKV: curves P-192, P-224, P-521 SIG(gen): curves P-192,P-224, P-521 SIG(ver): curves P-192,P-224 P-521	X	X	R W
Diffie-Hellman Key Agreement Key Size: 1024 bits	X	X	R W
Ed 25519 Key agreement, Signature Generation, Signature Verification	X	X	R W
SP800-56C Key Derivation Function	X	X	R W
CMAC AES 128 MAC Generation	X	X	R W
OMAC MAC Generation	X	X	R W
HMAC-DRGB Random Number Generation	X	X	R W
Hash-DRBG Random Number Generation	X	X	R W

Table 6b – Non-Approved Services in Non-Approved Mode

4.3 Operator authentication

Within the constraints of FIPS 140-2 level 1, the module does not implement an authentication mechanism for operator authentication. The assumption of a role is implicit in the action taken.

The module relies upon the operating system for any operator authentication.

5 Physical Security

The Apple iOS CoreCrypto Module, v5.0 is intended to operate on a multi-chip standalone platform used as a mobile device. The mobile device is comprised of production grade components and a production grade enclosure.

6 Operational Environment

The following sections describe the operational environment of the Apple iOS CoreCrypto Module, v5.0.

6.1 Applicability

The Apple iOS CoreCrypto Module, v5.0 operates in a modifiable operational environment per FIPS 140-2 level 1 specifications. It is part of iOS 8.0, a commercially available general-purpose operating system executing on the hardware specified in section 2.1.3.

6.2 Policy

The operating system is restricted to a single operator (single-user mode; i.e. concurrent operators are explicitly excluded).

When the operating system loads the module into memory, it invokes the FIPS Self-Test functionality, which in turn runs the mandatory FIPS 140-2 tests.

7 Cryptographic Key Management

The following section defines the key management features available through the Apple iOS CoreCrypto Module, v5.0.

7.1 Random Number Generation

A FIPS 140-2 approved deterministic random bit generator based on a block cipher as specified in NIST SP 800-90A is used. It is a CTR_DRBG using AES-128 with derivation function and without prediction resistance. The deterministic random bit generator is seeded by /dev/random. The /dev/random generator is a true random number generator that obtains entropy from interrupts generated by the devices and sensors attached to the system and maintains an entropy pool. The TRNG feeds entropy from the pool into the DRBG on demand. The TRNG provides 160-bits of entropy.

7.2 Key / CSP Generation

The following approved key generation methods are used by the module:

- The Approved DRBG specified in section 7.1 is used to generate cryptographic secret keys for symmetric key algorithms (AES, Triple-DES) and Message authentication (HMAC).
- The module provides PBKDF-based key generation services in the Approved mode.
- The Approved DRBG specified in section 7.1 is used to generate secret asymmetric keys for the ECDSA and RSA algorithm.

It is not possible for the module to output information during the key generating process. The DRBG itself is single-threaded.

The cryptographic strength of the 192 and 256 bit AES keys as well as the ECDSA keys for the curve P-384, as modified by the available entropy, is limited to 160-bits.

7.3 Key / CSP Establishment

The module provides RSA key wrapping, Diffie-Hellman- and EC Diffie-Hellman-based key establishment services.

The module provides key establishment services in the Approved mode through the PBKDFv2 algorithm. The PBKDFv2 function is provided as a service and returns the key derived from the provided password to the caller. The caller shall observe all requirements and should consider all recommendations specified in SP800-132 with respect to the strength of the generated key, including the quality of the password, the quality of the salt as well as the number of iterations. The implementation of the PBKDFv2 function requires the user to provide this information.

7.4 Key / CSP Entry and Output

All keys are imported from, or output to, the invoking application running on the same device. All keys entered into the module are electronically entered in plain text form. Keys are output from the module in plain text form if required by the calling application. The same holds for the CSPs.

7.5 Key / CSP Storage

The Apple iOS CoreCrypto Module, v5.0 considers all keys in memory to be ephemeral. They are received for use or generated by the module only at the command of the calling kernel service. The same holds for CSPs.

The module protects all keys, secret or private, and CSPs through the memory protection mechanisms provided by iOS. No process can read the memory of another process.

7.6 Key / CSP Zeroization

Keys and CSPs are zerorized when the appropriate context object is destroyed or when the device is powered down. Additionally, the user can zeroize the entire device directly (locally) or remotely, returning it to the original factory settings.

8 Electromagnetic Interference/Electromagnetic Compatibility (EMI/EMC)

The EMI/EMC properties of the CoreCrypto KEXT are not meaningful for the software library. The devices containing the software components of the module have their own overall EMI/EMC rating. The validation test environments have FCC, part 15, Class B rating.

9 Self-Tests

FIPS 140-2 requires that the module perform self-tests to ensure the integrity of the module and the correctness of the cryptographic functionality at start up. In addition, the random bit generator requires continuous verification. The FIPS Self Tests application runs all required module self-tests. This application is invoked by the iOS startup process upon device power on.

The execution of an independent application for invoking the self-tests in the libcorecrypto.dylib makes use of features of the iOS architecture: the module, implemented in libcorecrypto.dylib, is linked by libcommoncrypto.dylib which is linked by libSystem.dylib. The libSystem.dylib is a library that must be loaded into every application for operation. The library is stored in the kernel cache and therefore is not available on the disk as directly visible files. iOS ensures that there is only one physical instance of the library and maps it to all application linking to that library. In this way the module always stays in memory. Therefore, the self-test during startup time is sufficient as it tests the module instance loaded in memory which is subsequently used by every application on iOS.

All self-tests performed by the module are listed and described in this section.

9.1 Power-Up Tests

The following tests are performed each time the Apple iOS CoreCrypto Module, v5.0 starts and must be completed successfully for the module to operate in the FIPS approved mode. If any of the following tests fails the device powers itself off. To rerun the self-tests on demand, the user must reboot the device.

9.1.1 Cryptographic Algorithm Tests

Algorithm	Modes	Test
Triple-DES	CBC	KAT (Known Answer Test) Separate encryption / decryption operations are performed
Generic-software implementation (non-optimized based on LibTomCrypt): AES-128, AES-192, AES-256	CBC, ECB, GCM	KAT Separate encryption / decryption operations are performed
Generic-software implementation (non-optimized based on Gladman): AES-128, AES-192, AES-256	CBC	KAT Separate encryption / decryption operations are performed
Optimized-software implementation: AES-128, AES-192, AES-256	CBC, ECB, GCM	KAT Separate encryption / decryption operations are performed
Hardware implementation: AES-128, AES-192, AES-256	CBC	KAT Separate encryption / decryption operations are performed
DRBG	N/A	KAT
SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	N/A	KAT

HMAC-SHA-1, HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512	N/A	KAT
RSA	SIG(ver), SIG(gen) Encrypt / decrypt	KAT, pair-wise consistency checks Separate encryption /decryption operations are performed
ECDSA	SIG(ver), SIG(gen)	pair-wise consistency checks

Table 7: Cryptographic Algorithm Tests

9.1.2 Software / Firmware Integrity Tests

A software integrity test is performed on the runtime image of the Apple iOS CoreCrypto Module, v5.0. The CoreCrypto's HMAC-SHA-256 is used as an Approved algorithm for the integrity test. If the test fails, then the device powers itself off.

9.1.3 Critical Function Tests

No other critical function test is performed on power up.

9.2 Conditional Tests

The following sections describe the conditional tests supported by the Apple iOS CoreCrypto Module, v5.0.

9.2.1 Continuous Random Number Generator Test

The Apple iOS CoreCrypto Module, v5.0 performs a continuous random number generator test, whenever CTR_DRBG is invoked.

9.2.2 Pair-wise Consistency Test

The Apple iOS CoreCrypto Module, v5.0 does generate asymmetric keys and performs all required pair-wise consistency tests, the encryption/decryption as well as signature verification tests, with the newly generated key pairs.

9.2.3 SP 800-90A Assurance Tests

The Apple iOS CoreCrypto Module, v5.0 performs a subset of the assurance tests as specified in section 11 of SP 800-90A, in particular it complies with the mandatory documentation requirements and performs know-answer tests and prediction resistance.

9.2.4 Critical Function Test

No other critical function test is performed conditionally.

10 Design Assurance

10.1 Configuration Management

Apple manages and records source code and associated documentation files by using the revision control system called “Git”.

The Apple module hardware data, which includes descriptions, parts data, part types, bills of materials, manufacturers, changes, history, and documentation are managed and recorded. Additionally, configuration management is provided for the module’s FIPS documentation.

The following naming/numbering convention for documentation is applied.

<evaluation>_<module>_<os>_<mode>_<doc name>_<doc version (##.##)>

Example: FIPS_CORECRYPTO_IOS_US_SECPOL_01.03

Document management utilities provide access control, versioning, and logging. Access to the Git repository (source tree) is granted or denied by the server administrator in accordance with company and team policy.

10.2 Delivery and Operation

The CoreCrypto is built into iOS. For additional assurance, it is digitally signed. The Approved mode is configured by default and cannot be changed by a user.

10.3 Development

The Apple crypto module (like any other Apple software) undergoes frequent builds utilizing a “train” philosophy. Source code is submitted to the Build and Integration group (B & I). B & I builds, integrates and does basic sanity checking on the operating systems and apps that they produce. Copies of older versions are archived offsite in underground granite vaults.

10.4 Guidance

The following guidance items are to be used for assistance in maintaining the module’s validated status while in use.

10.4.1 Cryptographic Officer Guidance

The Approved mode of operation is configured in the system by default and cannot be changed. If the device starts up successfully then CoreCrypto has passed all self-tests and is operating in the Approved mode.

10.4.2 User Guidance

As above, the Approved mode of operation is configured in the system by default and cannot be changed. If the device starts up successfully then CoreCrypto has passed all self-tests and is operating in the Approved mode.

11 Mitigation of Other Attacks

The module protects against the utilization of known Triple-DES weak keys. The following keys are not permitted:

```
{0x01,0x01,0x01,0x01,0x01,0x01,0x01,0x01},
{0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE},
{0x1F,0x1F,0x1F,0x1F,0x0E,0x0E,0x0E,0x0E},
{0xE0,0xE0,0xE0,0xE0,0xF1,0xF1,0xF1,0xF1},
{0x01,0xFE,0x01,0xFE,0x01,0xFE,0x01,0xFE},
{0xFE,0x01,0xFE,0x01,0xFE,0x01,0xFE,0x01},
{0x1F,0xE0,0x1F,0xE0,0x0E,0xF1,0x0E,0xF1},
{0xE0,0x1F,0xE0,0x1F,0xF1,0x0E,0xF1,0x0E},
{0x01,0xE0,0x01,0xE0,0x01,0xF1,0x01,0xF1},
{0xE0,0x01,0xE0,0x01,0xF1,0x01,0xF1,0x01},
{0x1F,0xFE,0x1F,0xFE,0x0E,0xFE,0x0E,0xFE},
{0xFE,0x1F,0xFE,0x1F,0xFE,0x0E,0xFE,0x0E},
{0x01,0x1F,0x01,0x1F,0x01,0x0E,0x01,0x0E},
{0x1F,0x01,0x1F,0x01,0x0E,0x01,0x0E,0x01},
{0xE0,0xFE,0xE0,0xFE,0xF1,0xFE,0xF1,0xFE},
{0xFE,0xE0,0xFE,0xE0,0xFE,0xF1,0xFE,0xF1}.
```