Apple Inc.

# Apple CoreCrypto Kernel Module v8.0 for ARM
# FIPS 140-2 Non-Proprietary Security Policy

June, 2018

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 CoreCrypto Kernel Module v8.0 for ARM. The module's version is v8.0. 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 CMVP website at https://csrc.nist.gov/projects/cryptographic-module-validation-program .

Throughout the document "Apple CoreCrypto Kernel Module v8.0 for ARM", "cryptographic module", "CoreCrypto KEXT" or "the module" are used interchangeably to refer to the Apple CoreCrypto Kernel Module v8.0 for ARM. Throughout the document "OS" refers to "iOS", "tvOS", "watchOS" and "iBridgeOS (15P2064)" unless specifically noted.

## 1.2 Document Organization / Copyright

This non-proprietary Security Policy document may be reproduced and distributed only in its original entirety without any revision, ©2018 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 the OS releases with their corresponding validated modules and Crypto Officer Role Guides refer to the Apple Knowledge Base Articles HT202739 - "Product security certifications, validations, and guidance for iOS" (https://support.apple.com/en-us/HT202739).

The Cryptographic Module Validation Program website (https://csrc.nist.gov/projects/cryptographic-module-validation-program) 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 140-2 IG NIST, "Implementation Guidance for FIPS PUB 140-2 and the Cryptographic Module Validation Program," December 6, 2017

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 186-4 Federal Information Processing Standards Publication 186-4, July 2013, Digital Signature Standard (DSS)

| | |
|---|---|
| FIPS 197 | Federal Information Processing Standards Publication 197, November 26, 2001 Advanced Encryption Standard (AES) |
| FIPS 198 | Federal Information Processing Standards Publication 198, July, 2008 The Keyed-Hash Message Authentication Code (HMAC) |
| iOS | What's New in iOS 11<br>https://developer.apple.com/ios/ |
| SEC | Security Overview<br>http://developer.apple.com/library/ios/#documentation/Security/Conceptual/Security_Overview/Introduction/Introduction.html |
| SP800-38 A | NIST Special Publication 800-38A, "Recommendation for Block Cipher Modes of Operation", December 2001 |
| SP800-38 A | NIST Special Publication 800-38A, "Recommendation for Block Cipher Modes of Operation", December 2001 |
| SP800-38 E | NIST Special Publication 800-38E, "Recommendation for Block Cipher Modes of Operation: The XTS-AES Mode for Confidentiality on Storage Devices", January 2010 |
| SP800-38 F | NIST Special Publication 800-38E, "Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping", December 2012 |
| SP800-57P1 | NIST Special Publication 800-57, "Recommendation for Key Management – Part 1: General (Revised)", July 2012 |
| SP800-67 | NIST Special Publication 800-67, "Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher", (Revised) January 2012 |
| SP800-90A | NIST Special Publication 800-90A, "Recommendation for Random Number Generation Using Deterministic Random Bit Generators (Revised)", January 2012 |
| SP800-132 | NIST Special Publication 800-132, "Recommendation for Password-Based Key Derivation", December 2010 |
| UG | User Guidance for iOS: https://support.apple.com/en-us/HT202739 |

## 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 |
| DRBG | Deterministic Random Bit Generator |
| ECB | Electronic Codebook mode of operation |
| ECC | Elliptic Curve Cryptography |
| ECDSA | DSA based on ECC |
| EMC | Electromagnetic Compatibility |
| EMI | Electromagnetic Interference |
| FIPS | Federal Information Processing Standard |
| GCM | Galois/Counter Mode |
| HMAC | Hash-Based Message Authentication Code |
| KAT | Known Answer Test |
| KEXT | Kernel extension |
| KDF | Key Derivation Function |
| KPI | Kernel Programming Interface |
| KS | Key Size (Length) |
| MAC | Message Authentication Code |
| NIST | National Institute of Standards and Technology |
| OFB | Output Feedback (mode of operation) |
| PBKDF | Password-based Key Derivation Function |
| PCT | Pair-wise Consistency Test |
| RNG | Random Number Generator |
| SHS | Secure Hash Standard |
| Triple-DES | Triple Data Encryption Standard |

# 2 Cryptographic Module Specification

## 2.1 Module Description

The Apple CoreCrypto Kernel Module v8.0 for ARM 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
- Message authentication
- Signature generation/verification
- Random number generation
- Key derivation
- Key generation

### 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 CoreCrypto Kernel Module v8.0 for ARM are listed in detail. There are no components excluded from the validation testing.

#### 2.1.2.1 Software components

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

The CoreCrypto KEXT is linked dynamically into the OS kernel.

  

#### 2.1.2.2 Hardware components

There are no hardware components within the cryptographic module boundary.

### 2.1.3 Tested Platforms

The module has been tested with and without PAA[1] on the following platforms:

| Manufacturer | Model | Operating System |
|---|---|---|
| Apple Inc. | iPhone 5S with Apple A7 CPU | iOS 11 |
| Apple Inc. | iPhone 6 with Apple A8 CPU (iPhone 6 and iPhone 6 Plus) | iOS 11 |
| Apple Inc. | iPhone 6S with Apple A9 CPU (iPhone 6S and iPhone 6S Plus) | iOS 11 |
| Apple Inc. | iPhone 7 with Apple A10 Fusion CPU (iPhone 7 and iPhone 7 Plus) | iOS 11 |
| Apple Inc. | iPhone 8 with Apple A11 Bionic CPU | iOS 11 |
| Apple Inc. | iPad Air 2 with Apple A8X CPU | iOS 11 |
| Apple Inc. | iPad Pro with Apple A9X CPU | iOS 11 |
| Apple Inc. | iPad Pro with Apple A10X Fusion CPU | iOS 11 |
| Apple Inc. | Apple TV 4K with Apple A10X Fusion CPU | tvOS 11 |
| Apple Inc. | Apple Watch Series 1 with Apple S1P CPU | watchOS 4 |
| Apple Inc. | Apple Watch Series 3 with Apple S3 CPU | watchOS 4 |
| Apple Inc. | iMac Pro with Apple T2 (iBridge 2,1) | iBridgeOS (15P2064) |

Table 2: Tested Platforms

## 2.2 Modes of operation

The Apple CoreCrypto Kernel Module v8.0 for ARM has an Approved and Non-Approved Mode of operation. The Approved Mode of operation is configured in the system by default. If the device starts up successfully then CoreCrypto KEXT 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 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.

Approved security functions are listed in Table 3. Column four (Algorithm Certificate Number) of Table 3 lists the validation numbers obtained from NIST based on the successful CAVP testing of the cryptographic algorithm implementations on the platforms referenced in Table 2.

[1] PAA provided here is the ARM NEON present in Apple A series processors.

Refer to https://csrc.nist.gov/projects/cryptographic-algorithm-validation-program for the current standards, test requirements, and special abbreviations used in the following table:

### 2.2.1 Approved Security Functions

| Cryptographic Function | Algorithm | Modes/Options | Algorithm Certificate Number |
|---|---|---|---|
| Random Number Generation; Symmetric key generation | [SP800-90] DRBG | HMAC_DRBG<br>Generic Software Implementation<br>HMAC-SHA-384<br>HMAC-SHA-512 | 1849, 1850, 1851, 1852, 1853, 1854, 1855, 1856, 1861, 1917, 1918, 1959 |
| | | HMAC_DRBG<br>Optimized Assembler Using VNG<br>HMAC-SHA-1 HMAC-SHA-384<br>HMAC-SHA-224 HMAC-SHA-512<br>HMAC-SHA-256 | 1736, 1744, 1745, 1747, 1748, 1749, 1750, 1751, 1863, 1921, 1922, 1961 |
| | | CTR_DRBG<br>AES Optimized Assembler Implementation | 1737, 1738, 1739, 1740, 1741, 1742, 1743, 1746, 1860, 1915, 1916, 1962 |
| | | CTR_DRBG<br>AES Optimized Assembler and VNG Implementation | 1752, 1753, 1754, 1755, 1756, 1757, 1760, 1763, 1862, 1919, 1920, 1960 |
| Symmetric Encryption and Decryption | [FIPS 197]<br>AES<br>SP800-38 A<br>SP800-38 E<br>SP800-38 F | Generic Software Implementation (based on LibTomCrypt) using Assembler Implementation of ECB:<br>CBC ECB<br>CFB8 KW<br>CFB128 OFB<br>CTR | 4916, 4917, 4918, 4919, 4920, 4921, 4922, 4923, 5039, 5125, 5126, 5187 |
| | | Optimized Assembler and VNG implementation:<br>CCM GCM<br>CTR KW<br>ECB | 4924, 4925, 4926, 4927, 4928, 4929, 4932, 4935, 5040, 5127, 5128, 5185 |
| | | Optimized Assembler Implementation with ARM PAA<br>CBC KW<br>CFB128 OFB<br>ECB XTS | 4906, 4907, 4908, 4909, 4910, 4911, 4912, 4913, 5041, 5129, 5130, 5186 |
| | [SP800-67]<br>Triple-DES<br>ANSIX9.52-1 998 | 3-Key Triple-DES<br>(All keys independent)<br>Modes:<br>ECB<br>CBC | 2589, 2590, 2595, 2596, 2597, 2598, 2599, 2600, 2602, 2628, 2629, 2637 |
| Digital Signature and Asymmetric Key Generation | FIPS186-4 RSA<br>PKCS #1.5 | Signature Verification<br>Key Sizes:<br>1024<br>2048<br>3072 | 2717, 2718, 2719, 2720, 2721, 2722, 2723, 2724, 2728, 2763, 2764, 2783 |

| Cryptographic Function | Algorithm | Modes/Options | Algorithm Certificate Number |
|---|---|---|---|
| | [FIPS 186-4] ECDSA<br>ANSI X9.62 | PKG: curves<br>P-224, P-256, P-384, P-521<br>PKV: curves<br>P-224, P-256, P-384, P-521<br>Signature Generation: curves<br>P-224, P-256, P-384, P-521<br>using (SHA-224, SHA-256, SHA384, SHA512)<br>Signature Verification: curves<br>P-224, P-256, P-384, P-521<br>using (SHA-1, SHA-224, SHA-256, SHA384, SHA512) | 1289, 1290, 1291, 1292, 1293, 1294, 1295, 1296, 1298, 1325, 1326, 1345 |
| Message Digest | [FIPS 180-4]<br>SHS | Generic Software Implementation<br>SHA-1 SHA-384<br>SHA-224 SHA-512<br>SHA-256 | 4095, 4096, 4097, 4098, 4099, 4100, 4101, 4102, 4106, 4151, 4152, 4189 |
| | | Optimized Assembler Implementation using VNG:<br>SHA-1 SHA-384<br>SHA-224 SHA-512<br>SHA-256 | 4013, 4014, 4015, 4016, 4017, 4018, 4019, 4020, 4107, 4153, 4154, 4190 |
| Keyed Hash | [FIPS 198]<br>HMAC | Generic Software Implementation<br>SHA-1 SHA-384<br>SHA-224 SHA-512<br>SHA-256 | 3350, 3351, 3352, 3353, 3354, 3355, 3356, 3357, 3361, 3405, 3406, 3441 |
| | | Optimized Assembler Implementation using VNG:<br>SHA-1 SHA-384<br>SHA-224 SHA-512<br>SHA-256 | 3273, 3274, 3275, 3276, 3277, 3278, 3279, 3280, 3362, 3407, 3408, 3442 |
| Key Derivation | [SP800-132]<br>PBKDF | Password Based Key Derivation using HMAC with SHA-1 or SHA-2 | Vendor Affirmed |
| RSA Key Wrapping | PKCS1-v1_5<br>PKCS#1 v2.1<br>OAEP | Key Wrapping<br>Modulus size: 2048-bits or 3072-bits | Non-Approved, but Allowed[2] |
| MD5 | Message Digest | Digest size 128 bit | Non-Approved, but Allowed:<br>Used as part of the TLS key establishment scheme only |
| NDRNG | Random Number Generation | N/A | Non-Approved, but Allowed: provided by the underlying operational environment |

Table 3: Approved, Allowed and Vendor Affirmed Security Functions

[2] RSA listed in Table 3 is used in for key establishment (key wrapping). Methodology provides 112 or 128 bits of encryption strength

CAVEAT: The module generates cryptographic keys whose strengths are modified by available entropy – 160-bits. The encryption strength for the AES Key Wrapping using 192 and 256 bit keys is limited to 160 bits due to the entropy of the seed source.

Note: PBKDFv2 is implemented to support all options specified in Section 5.4 of SP800-132. The password consists of at least 6 alphanumeric characters from the ninety-six (96) printable and human-readable characters. The probability that a random attempt at guessing the password will succeed or a false acceptance will occur is equal to 1/96^6. The derived keys may only be used in storage applications. Additional guidance to appropriate usage is specified in section 7.3.

### 2.2.2 Non-Approved Security Functions

| Cryptographic Function | Usage / Description | Note |
|---|---|---|
| DES | Encryption / Decryption:<br>Key Size 56 bits | Non-Approved |
| MD2 | Message Digest<br>Digest size 128 bit | Non-Approved |
| MD4 | Message Digest<br>Digest size 128 bit | Non-Approved |
| RIPEMD | Message Digest<br>Digest size 128, 160, 256, 320 | Non-Approved |
| Ed25519 | Key Agreement<br>Signature Generation<br>Signature Verification | Non-Approved |
| ANSI X9.63 KDF | ANSI X9.63 Hash Based KDF | Non-Approved |
| RFC6637 KDF | KDF based on RFC6637 | Non-Approved |
| ECDSA | Key Pair Generation for compact point representation of points | Non-Approved |
| Integrated Encryption Scheme on elliptic curves | Encryption / Decryption | Non-Approved |
| ECDSA | PKG: Curve P-192<br>PKV: Curve P-192<br>Signature Generation: Curve P-192 | Non-Approved |
| CAST5 | Encryption / Decryption<br>Key Sizes 40 to 128 bits in 8-bit increments | Non-Approved |
| Blowfish | Encryption / Decryption | Non-Approved |
| RC2 | Encryption / Decryption | Non-Approved |
| RC4 | Encryption / Decryption | Non-Approved |
| OMAC (One-Key CBC MAC) | MAC generation | Non-Approved |
| RSA Key Wrapping | PKCS#1 v1.5<br>Key Size < 2048 | Non-Approved |

| Cryptographic Function | Usage / Description | Note |
|---|---|---|
| RSA Signature Verification | PKCS#1 v1.5<br>Key sizes (modulus): 1536 bits | Non-Compliant |
| SP800-108 | Key-Based KDF (KBKDF)<br>Modes: CTR, Feedback | Non-Compliant |
| SP800-56C | KDF | Non-Compliant |
| Triple-DES<br>Symmetric Encryption and Decryption | Optimized Assembler Implementation:<br>Encryption / Decryption<br>Mode: CTR | Non-Compliant |
| | Encryption / Decryption:<br>Two-Key implementation | |
| AES-CMAC | AES-128 MAC generation | Non-Compliant |

Table 4: Non-Approved or Non-Compliant Security Functions

Note: A Non-Approved function in Table 4 is that the function implements a non-Approved algorithm, while a Non-Compliant function is that the function implements an Approved algorithm but the implementation is not validated by the CAVP. Neither a Non-Compliant nor a Non-Approved function may be used in the Approved mode unless stated in the caveat found in Table 4.

## 2.3 Cryptographic Module Boundary

The physical boundary of the module is the physical boundary of the OS device (i.e. iPhone, iPad, Apple TV, Apple Watch, iMac Pro) 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.

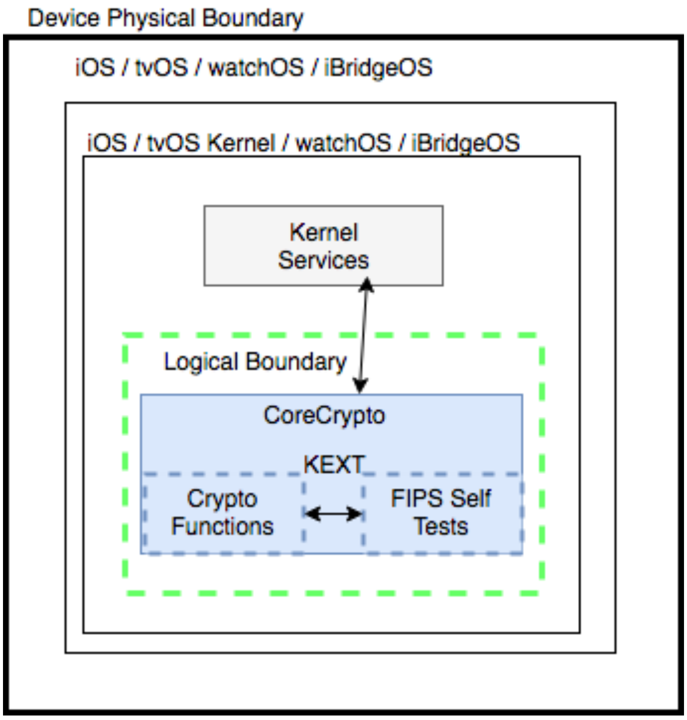


Figure 1: Logical Block Diagram

## 2.4 Module Usage Considerations

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

- In order to meet the IG A.13 requirement, the same Triple-DES key shall not be used to encrypt more than 2^28 64-bit blocks of data.
- 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, the caller must obtain a reference to the cipher implementation via the functions ccsha[1|224|256|384|512]_di.

# 3 Cryptographic Module Ports and Interfaces

The underlying logical interfaces of the module are the C Language Kernel Programming Interfaces (KPIs). In detail, these interfaces are the following:

- Data input and data output are provided in the variables passed in the KPI and callable service invocations, generally through caller-supplied buffers. Hereafter, KPIs and callable services will be referred to as “KPI.”
- Control inputs which control the mode of the module are provided through dedicated parameters, namely the kernel module plist whose information is supplied to the module by the kernel module loader.
- Status output is provided in return codes and through messages. Documentation for each KPI lists possible return codes. A complete list of all return codes returned by the C language KPIs within the module is provided in the header files and the KPI documentation. Messages are documented also in the KPI documentation.

The module is a kernel extension optimized for library use within the OS kernel and does not contain any terminating assertions or exceptions. Once the module is loaded into the OS kernel its cryptographic functions are made available to OS kernel services only. Any internal error detected by the module is reflected back to the caller with an appropriate return code. The calling OS Kernel service must examine the return code and act accordingly. There are two notable exceptions: (i) ECDSA does 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 error status synchronously through the use of documented return codes (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 listed in sections 2.1 and 4.2 |
| Crypto Officer (CO) | Utilization of services of the module listed in sections 2.1 and 4.2 |

Table 5: Roles

## 4.2 Services

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

Table 6 contains the cryptographic functions employed by the module in the 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 (secret and private cryptographic keys, for example) 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<br>*Input:* plaintext, IV, key<br>*Output:* ciphertext<br><br>Decryption<br>*Input:* ciphertext, IV, key<br>*Output:* plaintext | X | X | Secret key | R |

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

| Service | Roles | | CSPs & crypto keys | Access Type |
|---|---|---|---|---|
| | USER | CO | | |
| ECDSA signature generation and verification<br><br>Signature generation<br>*Input:* message m,<br>q, a, b, $X_G$, $Y_G$, n,<br>the SHA algorithm (SHA -224/SHA-256/SHA-384/SHA-512)<br>sender's private key d<br>*Output:* signature of m as a pair of r and s<br><br>Signature verification<br>*Input:* received message m',<br>signature in form on r' and s' pair,<br>q, a, b, $X_G$, $Y_G$, n,<br>sender's public key Q,<br>the SHA algorithm (SHA-1/SHA -224/SHA-256/SHA-384/SHA-512)<br>*Output:* pass if the signature is valid, fail if the signature is invalid | X | X | ECDSA key pair | R<br>W |
| Random number generation<br>*Input:* Entropy Input, Nonce, Personalization String<br>*Output:* Returned Bits | X | X | Entropy input string, Nonce, V and K | R<br>W<br>Z |
| ECDSA (key pair generation)<br>Input: q, FR, a, b, domain_parameter_seed, G, n, h.<br>Output: private key d, public key Q | X | X | Asymmetric key pair | R<br>W<br>Z |
| PBKDF<br>*Input:* encrypted key and password<br>*Output:* plaintext key<br>or<br>*Input:* plaintext key and password<br>*Output:* encrypted data | X | X | Secret key, password | R<br>W<br>Z |
| Release all resources of symmetric crypto function context<br>*Input:* context<br>*Output:* N/A | X | X | AES / Triple-DES key | Z |

| **Service** | **Roles** | | **CSPs & crypto keys** | **Access Type** |
|---|---|---|---|---|
| | USER | CO | | |
| Release all resources of hash context<br>*Input:* context<br>*Output:* N/A | X | X | HMAC key | Z |
| Release all resources of asymmetric crypto function context<br>*Input:* context<br>*Output:* N/A | X | X | Asymmetric keys (ECDSA) | 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 | |
| Integrated Encryption Scheme on elliptic curves encryption and decryption | X | X | R |
| DES Encryption and Decryption | X | X | R |
| Triple-DES (Optimized Assembler Implementation)<br>Encryption and Decryption<br>Mode: CTR | X | X | R |
| Triple-DES (Two-Key implementation) Encryption and Decryption | X | X | R |
| CAST5 Encryption and Decryption | X | X | R |
| Blowfish Encryption and Decryption | X | X | R |
| RC4 Encryption and Decryption | X | X | R |
| RC2 Encryption and Decryption | X | X | R |
| MD2 Hash | X | X | R<br>W |
| MD4 Hash | X | X | R<br>W |
| RIPEMD Hash | X | X | R<br>W |
| RSA PKCS#1 v1.5 Key Wrapping<br>Key sizes < 2048 | X | X | R<br>W |
| RSA PKCS#1 v1.5 Signature Verification<br>Key sizes: 1536 bits | X | X | R<br>W |
| ECDSA Key Pair Generation for compact point representation of points | X | X | R<br>W |

| Service | Roles | | Access Type |
|---|---|---|---|
| | USER | CO | |
| ECDSA<br>PKG: Curves P-192<br>PKV: Curves P-192<br>Signature Generation: Curves P-192<br>Signature Verification: Curves P-192 | X | X | R<br>W |
| Ed 25519 Key agreement, Signature Generation, Signature Verification | X | X | R<br>W |
| SP800-56C Key Derivation Function | X | X | R<br>W |
| ANSI X9.63 Key Derivation Function | X | X | R<br>W |
| SP800-108 Key Derivation Function<br>Modes: Feedback, Counter | X | X | R<br>W |
| RFC6637 Key Derivation Function | X | X | R<br>W |
| AES-CMAC MAC Generation | X | X | R<br>W |
| OMAC MAC Generation | X | X | R<br>W |

Table 7: 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 CoreCrypto Kernel Module v8.0 for ARM is intended to operate on a multi-chip standalone platform. The 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 CoreCrypto Kernel Module v8.0 for ARM.

## 6.1 Applicability

The Apple CoreCrypto Kernel Module v8.0 for ARM operates in a modifiable operational environment per FIPS 140-2 level 1 specifications. The module is included in the OS 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; concurrent operators are explicitly excluded).

FIPS Self-Test functionality is invoked along with mandatory FIPS 140-2 tests when the module is loaded into memory by the operating system.

  

# 7 Cryptographic Key Management

The following section defines the key management features available through the Apple CoreCrypto Kernel Module v8.0 for ARM.

## 7.1 Random Number Generation

The module uses a FIPS 140-2 approved deterministic random bit generator (DRBG) based on a block cipher as specified in NIST SP 800-90A. The default Approved DRBG used for random number generation is a CTR_DRBG using AES-128 with derivation function and without prediction resistance. The module also employs a HMAC-DRBG for random number generation. Seeding is obtained by read_random (a true random number generator). read_random 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 default Approved DRBG specified in section 7.1 is used to generate secret asymmetric key pairs for the ECDSA algorithm.

The module does not output any information or intermediate results during the key generation 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 key establishment services in the Approved Mode through the AES Key wrapping and PBKDFv2 algorithm. The RSA key wrapping is non-approved but allowed. 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, 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 entered from, or output to, the invoking kernel service 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 kernel service. The same holds for the CSPs.

## 7.5 Key / CSP Storage

The Apple CoreCrypto Kernel Module v8.0 for ARM 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 the OS, including the separation between the kernel and user-space. No process can read the memory of another process. No user-space application can read the kernel memory.

## 7.6 Key / CSP Zeroization

Keys and CSPs are zeroized 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 Apple CoreCrypto Kernel Module v8.0 for ARM 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 DRBG requires continuous verification. The FIPS Self Tests functionality runs all required module self-tests. This functionality is invoked by the OS Kernel startup process upon device initialization. If the self-tests succeed, the Apple CoreCrypto Kernel Module v8.0 for ARM instance is maintained in the memory of the OS Kernel on the device and made available to each calling kernel service without reloading. 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 CoreCrypto Kernel Module v8.0 for ARM 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 shuts down automatically. To run the self-tests on demand, the user may reboot the device.

### 9.1.1 Cryptographic Algorithm Tests

| Algorithm | Modes | Test |
|---|---|---|
| Triple-DES | CBC | KAT (Known Answer Test)<br>Separate encryption / decryption operations are performed |
| AES implementations selected by the module for the corresponding environment<br>AES-128 | ECB, CBC, XTS | KAT<br>Separate encryption / decryption operations are performed |
| DRBG (CTR_DRBG and HMAC_DRBG; tested separately) | N/A | KAT |
| HMAC-SHA-1, HMAC-SHA-256, HMAC-SHA-512 | N/A | KAT |
| ECDSA | Signature Generation, Signature Verification | pair-wise consistency test |
| RSA | Signature Verification | KAT |

Table 8: Cryptographic Algorithm Tests

### 9.1.2 Software / firmware integrity tests

A software integrity test is performed on the runtime image of the Apple CoreCrypto Kernel Module v8.0 for ARM. The CoreCrypto's HMAC-SHA256 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 CoreCrypto Kernel Module v8.0 for ARM.

### 9.2.1 Continuous Random Number Generator Test

The Apple CoreCrypto Kernel Module v8.0 for ARM performs a continuous random number generator test, whenever the DRBG is invoked.

  

In addition, the seed source implemented in the operating system kernel also performs a continuous self-test.

### 9.2.2 Pair-wise Consistency Test

The Apple CoreCrypto Kernel Module v8.0 for ARM generates asymmetric ECDSA key pairs and performs all required pair-wise consistency tests (signature generation and verification) with the newly generated key pairs.

### 9.2.3 SP800-90A Assurance Tests

The Apple CoreCrypto Kernel Module v8.0 for ARM performs a subset of the assurance tests as specified in section 11 of SP800-90A, in particular it complies with the mandatory documentation requirements and performs know-answer tests.

### 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 named “Git.”

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_tvOS_KS_SECPOL_4.0

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 KEXT is built into the OS. For additional assurance, it is digitally signed. The Approved Mode is configured by default and can only be transitioned into the non-Approved mode by calling one of the non-Approved algorithms listed in Table 4.

## 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 can only be transitioned into the non-Approved mode by calling one of the non-Approved algorithms listed in Table 4. If the device starts up successfully then CoreCrypto KEXT 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 can only be transitioned into the non-Approved mode by calling one of the non-Approved algorithms listed in Table 4. If the device starts up successfully then CoreCrypto KEXT has passed all self-tests and is operating in the Approved Mode.

Kernel programmers that use the module API shall not attempt to invoke any API call directly and only adhere to defined interfaces through the kernel framework.

  

# 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}.

  