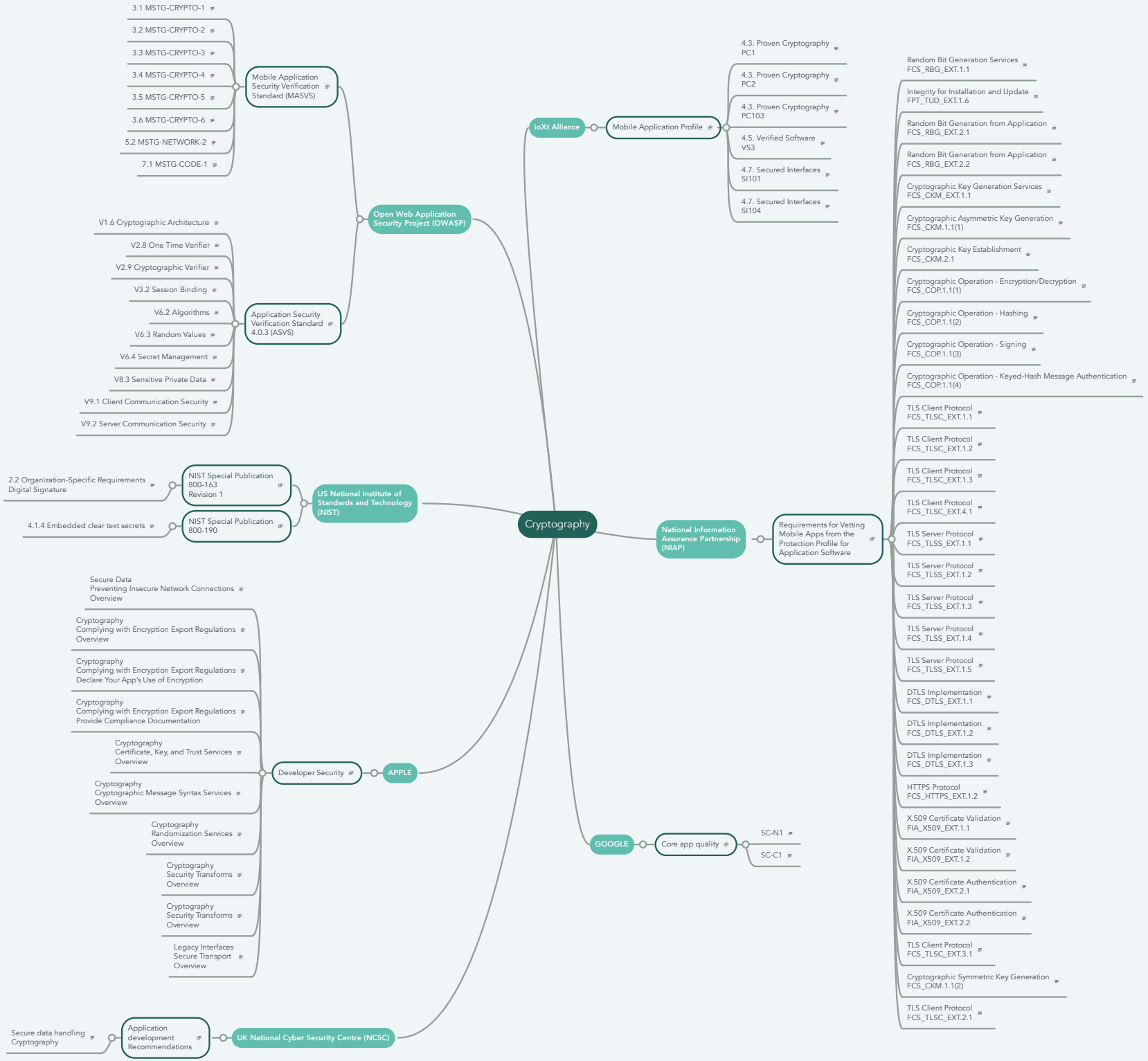
Cryptography
Open Web Application Security Project (OWASP)
Mobile Application Security Verification Standard (MASVS)
3.1 MSTG-CRYPTO-1
3.2 MSTG-CRYPTO-2
3.3 MSTG-CRYPTO-3
3.4 MSTG-CRYPTO-4
3.5 MSTG-CRYPTO-5
3.6 MSTG-CRYPTO-6
5.2 MSTG-NETWORK-2
7.1 MSTG-CODE-1
Application Security Verification Standard 4.0.3 (ASVS)
V1.6 Cryptographic Architecture
V2.8 One Time Verifier
V2.9 Cryptographic Verifier
V3.2 Session Binding
V6.2 Algorithms
V6.3 Random Values
V6.4 Secret Management
V8.3 Sensitive Private Data
V9.1 Client Communication Security
V9.2 Server Communication Security
US National Institute of Standards and Technology (NIST)
NIST Special Publication 800-163 Revision 1
2.2 Organization-Specific Requirements Digital Signature
NIST Special Publication 800-190
4.1.4 Embedded clear text secrets
APPLE
Developer Security
Secure Data
Preventing Insecure Network Connections
Overview
Cryptography
Complying with Encryption Export Regulations
Overview
Cryptography
Complying with Encryption Export Regulations
Declare Your App's Use of Encryption
Cryptography
Complying with Encryption Export Regulations
Provide Compliance Documentation
Cryptography
Certificate, Key, and Trust Services
Overview
Cryptography
Cryptographic Message Syntax Services
Overview
Cryptography
Randomization Services
Overview
Cryptography
Security Transforms
Overview
Cryptography
Security Transforms
Overview
Legacy Interfaces
Secure Transport
Overview
UK National Cyber Security Centre (NCSC)
Application development Recommendations
Secure data handling
Cryptography
ioXt Alliance
Mobile Application Profile
4.3. Proven Cryptography PC1
4.3. Proven Cryptography PC2
4.3. Proven Cryptography PC103
4.5. Verified Software VS3
4.7. Secured Interfaces SI101
4.7. Secured Interfaces SI104
National Information Assurance Partnership (NIAP)
Requirements for Vetting Mobile Apps from the Protection Profile for Application Software
Random Bit Generation Services
FCS_RBG_EXT.1.1
Integrity for Installation and Update
FPT_TUD_EXT.1.6
Random Bit Generation from Application
FCS_RBG_EXT.2.1
Random Bit Generation from Application
FCS_RBG_EXT.2.2
Cryptographic Key Generation Services
FCS_CKM_EXT.1.1
Cryptographic Asymmetric Key Generation
FCS_CKM.1.1(1)
Cryptographic Key Establishment
FCS_CKM.2.1
Cryptographic Operation - Encryption/Decryption
FCS_COP.1.1(1)
Cryptographic Operation - Hashing
FCS_COP.1.1(2)
Cryptographic Operation - Signing
FCS_COP.1.1(3)
Cryptographic Operation - Keyed-Hash Message Authentication
FCS_COP.1.1(4)
TLS Client Protocol
FCS_TLSC_EXT.1.1
TLS Client Protocol
FCS_TLSC_EXT.1.2
TLS Client Protocol
FCS_TLSC_EXT.1.3
TLS Client Protocol
FCS_TLSC_EXT.4.1
TLS Server Protocol
FCS_TLSS_EXT.1.1
TLS Server Protocol
FCS_TLSS_EXT.1.2
TLS Server Protocol
FCS_TLSS_EXT.1.3
TLS Server Protocol
FCS_TLSS_EXT.1.4
TLS Server Protocol
FCS_TLSS_EXT.1.5
DTLS Implementation
FCS_DTLS_EXT.1.1
DTLS Implementation
FCS_DTLS_EXT.1.2
DTLS Implementation
FCS_DTLS_EXT.1.3
HTTPS Protocol
FCS_HTTPS_EXT.1.2
X.509 Certificate Validation
FIA_X509_EXT.1.1
X.509 Certificate Validation
FIA_X509_EXT.1.2
X.509 Certificate Authentication
FIA_X509_EXT.2.1
X.509 Certificate Authentication
FIA_X509_EXT.2.2
TLS Client Protocol
FCS_TLSC_EXT.3.1
Cryptographic Symmetric Key Generation
FCS_CKM.1.1(2)
TLS Client Protocol
FCS_TLSC_EXT.2.1
GOOGLE
Core app quality
SC-N1
SC-C1

Cryptography

1. National Information Assurance Partnership (NIAP)

1.1. Requirements for Vetting Mobile Apps from the Protection Profile for Application Software

Link: https://www.niap-ccevs.org/MMO/PP/394.R/pp_app_v1.2_table-reqs.htm

1.1.1. Random Bit Generation Services FCS_RBG_EXT.1.1

> The application shall [selection:
> use no DRBG functionality, invoke platform-provided DRBG functionality, implement DRBG functionality] for its cryptographic operations.
> Application Note: If implement DRBG functionality is chosen, then additional FCS_RBG_EXT.2 elements shall be included in the ST. In this requirement, cryptographic operations include all cryptographic key generation/derivation/agreement, IVs (for certain modes), as well as protocol-specific random values.

1.1.2. Integrity for Installation and Update FPT_TUD_EXT.1.6

> The application installation package and its updates shall be digitally signed such that its platform can cryptographically verify them prior to installation.
> Application Note: The specifics of the verification of installation packages and updates involves requirements on the platform (and not the application), so these are not fully specified here.

1.1.3. Random Bit Generation from Application FCS_RBG_EXT.2.1

> The application shall perform all deterministic random bit generation (DRBG) services in accordance with NIST Special Publication 800-90A using [selection: Hash_DRBG (any), HMAC_DRBG (any), CTR_DRBG (AES)] .
> Application Note: This requirement shall be included in STs in which implement DRBG functionality is chosen in FCS_RBG_EXT.1.1. The ST author should select the standard to which the RBG services comply (either SP 800-90A or FIPS 140-2 Annex C).
> SP 800-90A contains three different methods of generating random numbers; each of these, in turn, depends on underlying cryptographic primitives (hash functions/ciphers). The ST author will select the function used (if SP 800-90A is selected), and include the specific underlying cryptographic primitives used in the requirement or in the TSS. While any of the identified hash functions (SHA-1, SHA-224, SHA-256, SHA-384, SHA-512) are allowed for Hash_DRBG or HMAC_DRBG, only AES-based implementations for CTR_DRBG are allowed.

1.1.4. Random Bit Generation from Application FCS_RBG_EXT.2.2

The deterministic RBG shall be seeded by an entropy source that accumulates entropy from a platform-based DRBG and [selection: a software-based noise source, no other noise source] with a minimum of [selection: 128 bits,
256 bits] of entropy at least equal to the greatest security strength (according to NIST SP 800-57) of the keys and hashes that it will generate.
Application Note: This requirement shall be included in STs in which implement DRBG functionality is chosen in FCS_RBG_EXT.1.1. For the first selection in this requirement, the ST author selects 'software-based noise source' if any additional noise sources are used as input to the application's DRBG. Note that the application must use the platform's DRBG to seed its DRBG.
In the second selection in this requirement, the ST author selects the appropriate number of bits of entropy that corresponds to the greatest security strength of the algorithms included in the ST. Security strength is defined in Tables 2 and 3 of NIST SP 800-57A. For example, if the implementation includes 2048-bit RSA (security strength of 112 bits) and AES 256 (security strength 256 bits), then the ST author would select 256 bits.

1.1.5. Cryptographic Key Generation Services FCS_CKM_EXT.1.1

The application shall [selection:
generate no asymmetric cryptographic keys, invoke platform-provided functionality for asymmetric key generation, implement asymmetric key generation] .
Application Note: If implement asymmetric key generation or invoke platform-provided functionality for asymmetric key generation is chosen, then additional FCS_CKM.1(1) elements shall be included in the ST.

1.1.6. Cryptographic Asymmetric Key Generation FCS_CKM.1.1(1)

The application shall generate asymmetric cryptographic keys in accordance with a specified cryptographic key generation algorithm [selection: [RSA schemes] using cryptographic key sizes of [2048-bit or greater] that meet the following: [selection:

FIPS PUB 186-4, “Digital Signature Standard (DSS)”, Appendix B.3 ,
ANSI X9.31-1998, Section 4.1] ,
[ECC schemes] using [“NIST curves” P-256, P-384 and [selection: P-521 , no other curves]] that meet the following: [FIPS PUB 186-4, “Digital Signature Standard (DSS)”, Appendix B.4] ,

[FFC schemes] using cryptographic key sizes of [2048-bit or greater] that meet the following: [FIPS PUB 186-4, “Digital Signature Standard (DSS)”, Appendix B.1]
] .Application Note: The ST author shall select all key generation schemes used for key establishment and entity authentication. When key generation is used for key establishment, the schemes in FCS_CKM.2.1 and selected cryptographic protocols must match the selection. When key generation is used for entity authentication, the public key is expected to be associated with an X.509v3 certificate.
If the TOE acts as a receiver in the RSA key establishment scheme, the TOE does not need to implement RSA key generation.
The ANSI X9.31-1998 option will be removed from the selection in a future publication of this document. Presently, the selection is not exclusively limited to the FIPS PUB 186-4 options in order to allow industry some further time to complete the transition to the modern FIPS PUB 186-4 standard.

1.1.7. Cryptographic Key Establishment FCS_CKM.2.1

The application shall [selection: invoke platform-provided functionality , implement functionality] to perform cryptographic key establishment in accordance with a specified cryptographic key establishment method:
[RSA-based key establishment schemes] that meets the following: [NIST Special Publication 800-56B, “Recommendation for Pair-Wise Key Establishment Schemes Using Integer Factorization Cryptography”]
and [selection:
[Elliptic curve-based key establishment schemes] that meets the following: [NIST Special Publication 800-56A, “Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography”] ,[Finite field-based key establishment schemes] that meets the following: [NIST Special Publication 800-56A, “Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography”] , No other schemes] .
Application Note: The ST author shall select all key establishment schemes used for the selected cryptographic protocols. FCS_TLSC_EXT.1 requires cipher suites that use RSA-based key establishment schemes.
The RSA-based key establishment schemes are described in Section 9 of NIST SP 800-56B; however, Section 9 relies on implementation of other sections in SP 800-56B. If the TOE acts as a receiver in the RSA key establishment scheme, the TOE does not need to implement RSA key generation.
The elliptic curves used for the key establishment scheme shall correlate with the curves specified in FCS_CKM.1.1(1).
The domain parameters used for the finite field-based key establishment scheme are specified by the key generation according to FCS_CKM.1.1(1).

1.1.8. Cryptographic Operation - Encryption/Decryption FCS_COP.1.1(1)

The application shall perform encryption/decryption in accordance with a specified cryptographic algorithm
AES-CBC (as defined in NIST SP 800-38A) mode; and [selection: AES-GCM (as defined in NIST SP 800-38D),
no other modes] and cryptographic key sizes 256-bit and [selection: 128-bit, no other key sizes] .
Application Note: For the first selection, the ST author should choose the mode or modes in which AES operates. For the second selection, the ST author should choose the key sizes that are supported by this functionality. 128-bit key size is required in order to comply with FCS_TLSC_EXT.1 and FCS_CKM.1(1), if those are selected.

1.1.9. Cryptographic Operation - Hashing FCS_COP.1.1(2)

The application shall perform cryptographic hashing services in accordance with a specified cryptographic algorithm [selection:
SHA-1, SHA-256, SHA-384, SHA-512, no other algorithms] and message digest sizes [selection: 160, 256, 384, 512, no other message digest sizes] bits that meet the following: FIPS Pub 180-4.
Application Note: Per NIST SP 800-131A, SHA-1 for generating digital signatures is no longer allowed, and SHA-1 for verification of digital signatures is strongly discouraged as there may be risk in accepting these signatures.
SHA-1 is currently required in order to comply with FCS_TLSC_EXT.1. If FCS_TLSC_EXT.1.1 is included in the ST, the hashing algorithms selection for FCS_COP.1(2) must match the hashing algorithms used in the mandatory and selected cipher suites of FCS_TLSC_EXT.1.1.
Vendors are strongly encouraged to implement updated protocols that support the SHA-2 family; until updated protocols are supported, this PP allows support for SHA-1 implementations in compliance with SP 800-131A.
The intent of this requirement is to specify the hashing function. The hash selection must support the message digest size selection. The hash selection should be consistent with the overall strength of the algorithm used (for example, SHA 256 for 128-bit keys).

1.1.10. Cryptographic Operation - Signing FCS_COP.1.1(3)

The application shall perform cryptographic signature services (generation and verification) in accordance with a specified cryptographic algorithm [selection:
RSA schemes using cryptographic key sizes of 2048-bit or greater that meet the following: FIPS PUB 186-4, “Digital Signature Standard (DSS)”, Section 4 ,
ECDSA schemes using “NIST curves” P-256, P-384 and [selection: P-521, no other curves] that meet the following: FIPS PUB 186-4, “Digital Signature Standard (DSS)”, Section 5] .
Application Note: The ST Author should choose the algorithm implemented to perform digital signatures; if more than one algorithm is available, this requirement should be iterated to specify the functionality. For the algorithm chosen, the ST author should make the appropriate assignments/selections to specify the parameters that are implemented for that algorithm. RSA signature generation and verification is currently required in order to comply with FCS_TLSC_EXT.1.

1.1.11. Cryptographic Operation - Keyed-Hash Message Authentication FCS_COP.1.1(4)

The application shall perform keyed-hash message authentication in accordance with a specified cryptographic algorithm HMAC-SHA-256 and [selection: SHA-1,
SHA-384, SHA-512, no other algorithms] with key sizes [assignment: key size (in bits) used in HMAC] and message digest sizes 256 and [selection: 160, 384, 512, no other size] bits that meet the following: FIPS Pub 198-1 The Keyed-Hash Message Authentication Code and FIPS Pub 180-4 Secure Hash Standard.
Application Note: The intent of this requirement is to specify the keyed-hash message authentication function used for key establishment purposes for the various cryptographic protocols used by the application (e.g., trusted channel). The hash selection must support the message digest size selection. The hash selection should be consistent with the overall strength of the algorithm used for FCS_COP.1(1). HMAC-SHA256 is required in order to comply with the required cipher suites in FCS_TLSC_EXT.1.

1.1.12. TLS Client Protocol FCS_TLSC_EXT.1.1

The application shall [selection: invoke platform-provided TLS 1.2, implement TLS 1.2 (RFC 5246)] supporting the following cipher suites:
Mandatory Cipher Suites: TLS_RSA_WITH_AES_128_CBC_SHA as defined in RFC 5246
Optional Cipher Suites: [selection:
TLS_DHE_RSA_WITH_AES_128_CBC_SHA256 as defined in RFC 5246,
TLS_DHE_RSA_WITH_AES_256_CBC_SHA256 as defined in RFC 5246,
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 as defined in RFC 5289,
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 as defined in RFC 5289,
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 as defined in RFC 5289,
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 as defined in RFC 5289,
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 as defined in RFC 5289,
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 as defined in RFC 5289,
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 as defined in RFC 5289,
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 as defined in RFC 5289,
TLS_RSA_WITH_AES_128_CBC_SHA256 as defined in RFC 5246,
TLS_RSA_WITH_AES_256_CBC_SHA256 as defined in RFC 5246,no other cipher suite] .
Application Note: The cipher suites to be tested in the evaluated configuration are limited by this requirement. The ST author should select the optional cipher suites that are supported; if there are no cipher suites supported other than the mandatory suites, then “None” should be selected. It is necessary to limit the cipher suites that can be used in an evaluated configuration administratively on the server in the test environment. The Suite B algorithms listed above (RFC 6460) are the preferred algorithms for implementation.
TLS_RSA_WITH_AES_128_CBC_SHA is required in order to ensure compliance with RFC 5246.
These requirements will be revisited as new TLS versions are standardized by the IETF.
If any cipher suites are selected using ECDHE, then FCS_TLSC_EXT.4 is required.
If implement TLS 1.2 (RFC 5246) is selected, then FCS_CKM.2, FCS_CKM_EXT.1, FCS_COP.1(1), FCS_COP.1(2), FCS_COP.1(3), and FCS_COP.1(4) are required.

1.1.13. TLS Client Protocol FCS_TLSC_EXT.1.2

The application shall verify that the presented identifier matches the reference identifier according to RFC 6125.
Application Note: The rules for verification of identity are described in Section 6 of RFC 6125. The reference identifier is established by the user (e.g. entering a URL into a web browser or clicking a link), by configuration (e.g. configuring the name of a mail server or authentication server), or by an application (e.g. a parameter of an API) depending on the application service. Based on a singular reference identifier's source domain and application service type (e.g. HTTP, SIP, LDAP), the client establishes all reference identifiers which are acceptable, such as a Common Name for the Subject Name field of the certificate and a (case-insensitive) DNS name, URI name, and Service Name for the Subject Alternative Name field. The client then compares this list of all acceptable reference identifiers to the presented identifiers in the TLS server's certificate.
The preferred method for verification is the Subject Alternative Name using DNS names, URI names, or Service Names. Verification using the Common Name is required for the purposes of backwards compatibility. Additionally, support for use of IP addresses in the Subject Name or Subject Alternative name is discouraged as against best practices but may be implemented. Finally, the client should avoid constructing reference identifiers using wildcards. However, if the presented identifiers include wildcards, the client must follow the best practices regarding matching; these best practices are captured in the assurance activity.

1.1.14. TLS Client Protocol FCS_TLSC_EXT.1.3

The application shall establish a trusted channel only if the peer certificate is valid.
Application Note: Validity is determined by the identifier verification, certificate path, the expiration date, and the revocation status in accordance with RFC 5280. Certificate validity shall be tested in accordance with testing performed for FIA_X509_EXT.1.
For TLS connections, this channel shall not be established if the peer certificate is invalid. The HTTPS protocol (FCS_HTTPS_EXT.1) requires different behavior, though HTTPS is implemented over TLS. This element addresses non-HTTPS TLS connections.

1.1.15. TLS Client Protocol FCS_TLSC_EXT.4.1

The application shall present the supported Elliptic Curves Extension in the Client Hello with the following NIST curves: [selection: secp256r1, secp384r1, secp521r1] and no other curves.
Application Note: This requirement limits the elliptic curves allowed for authentication and key agreement to the NIST curves from FCS_COP.1(3) and FCS_CKM.1(1) and FCS_CKM.2. This extension is required for clients supporting Elliptic Curve cipher suites.

1.1.16. TLS Server Protocol FCS_TLSS_EXT.1.1

The application shall [selection: invoke platform-provided TLS 1.2, implement TLS 1.2 (RFC 5246)] supporting the following cipher suites:
Mandatory Cipher Suites: TLS_RSA_WITH_AES_128_CBC_SHA as defined in RFC 5246
Optional Cipher Suites: [selection:
TLS_DHE_RSA_WITH_AES_128_CBC_SHA256 as defined in RFC 5246,
TLS_DHE_RSA_WITH_AES_256_CBC_SHA256 as defined in RFC 5246,
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 as defined in RFC 5289,
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 as defined in RFC 5289,
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 as defined in RFC 5289,
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 as defined in RFC 5289,
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 as defined in RFC 5289,
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 as defined in RFC 5289,
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 as defined in RFC 5289,
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 as defined in RFC 5289,
TLS_RSA_WITH_AES_128_CBC_SHA256 as defined in RFC 5246,
TLS_RSA_WITH_AES_256_CBC_SHA256 as defined in RFC 5246,
no other cipher suite] .
Application Note: The cipher suites to be tested in the evaluated configuration are limited by this requirement. The ST author should select the optional cipher suites that are supported; if there are no cipher suites supported other than the mandatory suites, then “None” should be selected. It is necessary to limit the cipher suites that can be used in an evaluated configuration administratively on the server in the test environment. The Suite B algorithms listed above (RFC 6460) are the preferred algorithms for implementation.
TLS_RSA_WITH_AES_128_CBC_SHA is required in order to ensure compliance with RFC 5246.
These requirements will be revisited as new TLS versions are standardized by the IETF.
If any cipher suites are selected using ECDHE, then FCS_TLSC_EXT.4 is required.
If implement TLS 1.2 (RFC 5246) is selected, then FCS_CKM.2.1, FCS_COP.1.1(1), FCS_COP.1.1(2), FCS_COP.1.1(3), and FCS_COP.1.1(4) are required.

1.1.17. TLS Server Protocol FCS_TLSS_EXT.1.2

The application shall deny connections from clients requesting SSL 2.0, SSL 3.0, TLS 1.0, TLS 1.1, and [selection: TLS 1.2, none] .
Application Note: All SSL versions and TLS 1.0 and 1.1 are denied. Any TLS version not selected in FCS_TLSS_EXT.1.1 should be selected here.

1.1.18. TLS Server Protocol FCS_TLSS_EXT.1.3

The application shall generate key establishment parameters using RSA with size 2048 bits and [selection: 3072 bits, 4096 bits, no other sizes] and [selection: over NIST curves [selection: secp256r, secp384r] and no other curves, Diffe-Hellman parameters of size 2048 and [selection: 3072 bits, no other size] , no other]
Application Note: If the ST lists a DHE ciphersuite in FCS_TLSS_EXT.1.1, the ST must include the Diffie-Hellman selection in the requirement

1.1.19. TLS Server Protocol FCS_TLSS_EXT.1.4

The application shall support mutual authentication of TLS clients using X.509v3 certificates.

1.1.20. TLS Server Protocol FCS_TLSS_EXT.1.5

The application shall not establish a trusted channel if the distinguished name (DN) or Subject Alternative Name (SAN) contained in a certificate does not match the expected identifier for the peer.
Application Note: The peer identifier may be in the Subject field or the Subject Alternative Name extension of the certificate. The expected identifier may either be configured, may be compared to the Domain Name, IP address, username, or email address used by the peer, or may be passed to a directory server for comparison. Matching should be performed by a bit-wise comparison.

1.1.21. DTLS Implementation FCS_DTLS_EXT.1.1

The application shall implement the DTLS protocol in accordance with DTLS 1.2 (RFC 6347).

1.1.22. DTLS Implementation FCS_DTLS_EXT.1.2

The application shall implement the requirements in TLS (FCS_TLSC_EXT.1) for the DTLS implementation, except where variations are allowed according to DTLS 1.2 (RFC 6347).
Application Note: Differences between DTLS 1.2 and TLS 1.2 are outlined in RFC 6347; otherwise the protocols are the same. In particular, for the applicable security characteristics defined for the TSF, the two protocols do not differ. Therefore, all application notes and assurance activities that are listed for TLS apply to the DTLS implementation.

1.1.23. DTLS Implementation FCS_DTLS_EXT.1.3

The application shall not establish a trusted communication channel if the peer certificate is deemed invalid.
Application Note: Validity is determined by the certificate path, the expiration date, and the revocation status in accordance with RFC 5280.

1.1.24. HTTPS Protocol FCS_HTTPS_EXT.1.2

The application shall implement HTTPS using TLS (FCS_TLSC_EXT.1).

1.1.25. X.509 Certificate Validation FIA_X509_EXT.1.1

The application shall [selection: invoked platform-provided functionality , implement functionality] to validate certificates in accordance with the following rules:
RFC 5280 certificate validation and certificate path validation.
The certificate path must terminate with a trusted CA certificate.
The application shall validate a certificate path by ensuring the presence of the basicConstraints extension and that the CA flag is set to TRUE for all CA certificates.
The application shall validate the revocation status of the certificate using [selection: the Online Certificate Status Protocol (OCSP) as specified in RFC 2560 , a Certificate Revocation List (CRL) as specified in RFC 5759 , an OCSP TLS Status Request Extension (i.e., OCSP stapling) as specified in RFC 6066] .
The application shall validate the extendedKeyUsage field according to the following rules:
Certificates used for trusted updates and executable code integrity verification shall have the Code Signing purpose (id-kp 3 with OID 1.3.6.1.5.5.7.3.3) in the extendedKeyUsage field.
Server certificates presented for TLS shall have the Server Authentication purpose (id-kp 1 with OID 1.3.6.1.5.5.7.3.1) in the extendedKeyUsage field.
Client certificates presented for TLS shall have the Client Authentication purpose (id-kp 2 with OID 1.3.6.1.5.5.7.3.2) in the extendedKeyUsage field.
S/MIME certificates presented for email encryption and signature shall have the Email Protection purpose (id-kp 4 with OID 1.3.6.1.5.5.7.3.4) in the extendedKeyUsage field.
OCSP certificates presented for OCSP responses shall have the OCSP Signing purpose (id-kp 9 with OID 1.3.6.1.5.5.7.3.9) in the extendedKeyUsage field.
Server certificates presented for EST shall have the CMC Registration Authority (RA) purpose (id-kp-cmcRA with OID 1.3.6.1.5.5.7.3.28) in the extendedKeyUsage field.

Application Note: FIA_X509_EXT.1.1 lists the rules for validating certificates. The ST author shall select whether revocation status is verified using OCSP or CRLs. FIA_X509_EXT.2 requires that certificates are used for HTTPS, TLS and DTLS; this use requires that the extendedKeyUsage rules are verified.
Regardless of the selection of implement functionality or invoke platform-provided functionality, the validation is expected to end in a trusted root CA certificate in a root store managed by the platform.

1.1.26. X.509 Certificate Validation FIA_X509_EXT.1.2

The application shall treat a certificate as a CA certificate only if the basicConstraints extension is present and the CA flag is set to TRUE.
Application Note: This requirement applies to certificates that are used and processed by the TSF and restricts the certificates that may be added as trusted CA certificates.

1.1.27. X.509 Certificate Authentication FIA_X509_EXT.2.1

The application shall use X.509v3 certificates as defined by RFC 5280 to support authentication for [selection: HTTPS , TLS , DTLS] .
Application Note: The ST author's selection shall match the selection in FTP_DIT_EXT.1.1.

1.1.28. X.509 Certificate Authentication FIA_X509_EXT.2.2

When the application cannot establish a connection to determine the validity of a certificate, the application shall [selection: allow the administrator to choose whether to accept the certificate in these cases , accept the certificate , not accept the certificate] .
Application Note: Often a connection must be established to perform a verification of the revocation status of a certificate - either to download a CRL or to perform OCSP. The selection is used to describe the behavior in the event that such a connection cannot be established (for example, due to a network error). If the TOE has determined the certificate valid according to all other rules in FIA_X509_EXT.1, the behavior indicated in the selection shall determine the validity. The TOE must not accept the certificate if it fails any of the other validation rules in FIA_X509_EXT.1.

1.1.29. TLS Client Protocol FCS_TLSC_EXT.3.1

The application shall present the signature_algorithms extension in the Client Hello with the supported_signature_algorithms value containing the following hash algorithms: [selection: SHA256, SHA384, SHA512] and no other hash algorithms.
Application Note: This requirement limits the hashing algorithms supported for the purpose of digital signature verification by the client and limits the server to the supported hashes for the purpose of digital signature generation by the server. The signature_algorithm extension is only supported by TLS 1.2.

1.1.30. Cryptographic Symmetric Key Generation FCS_CKM.1.1(2)

The application shall generate symmetric cryptographic keys using a Random Bit Generator as specified in FCS_RBG_EXT.1 and specified cryptographic key sizes [selection:
128 bit,256 bit] .
Application Note: Symmetric keys may be used to generate keys along the key chain.

1.1.31. TLS Client Protocol FCS_TLSC_EXT.2.1

The application shall support mutual authentication using X.509v3 certificates.
Application Note: The use of X.509v3 certificates for TLS is addressed in FIA_X509_EXT.2.1. This requirement adds that a client must be capable of presenting a certificate to a TLS server for TLS mutual authentication.

2. GOOGLE

2.1. Core app quality

Link: https://developer.android.com/docs/quality-guidelines/core-app-quality

2.1.1. SC-N1

All network traffic is sent over SSL.

2.1.2. SC-C1

The app uses strong, platform-provided cryptographic algorithms and a random number generator. Also, the app does not implement custom algorithms.

3. UK National Cyber Security Centre (NCSC)

3.1. Application development Recommendations

Link: https://www.ncsc.gov.uk/collection/application-development/generic-application-development

3.1.1. Secure data handling Cryptography

Secure implementation of cryptographic functions requires significant effort to properly design and verify, so wherever possible use native capabilities available on the platform.
If using non-native cryptographic schemes, ensure they are reviewed and tested by third party experts.
Storing cryptographic keys on the device will reduce the effectiveness of an additional cryptographic layer (as keys stored locally could be recovered from the device). Storing the keys on a remote server would prevent an attacker with physical access to the device from retrieving them. Users should be required to authenticate to the server. Alternately, you can use TPMs (trusted platform modules) and secure enclaves to improve the security of stored keys.

4. APPLE

4.1. Developer Security

Link: https://developer.apple.com/documentation/security

4.1.1. Secure Data Preventing Insecure Network Connections Overview

On Apple platforms, a networking security feature called App Transport Security (ATS) improves privacy and data integrity for all apps and app extensions. It does this by requiring that network connections made by your app are secured by the Transport Layer Security (TLS) protocol using reliable certificates and ciphers. ATS blocks connections that don't meet minimum security requirements.

4.1.2. Cryptography Complying with Encryption Export Regulations Overview

When you submit your app to TestFlight or the App Store, you upload your app to a server in the United States. If you distribute your app outside the U.S. or Canada, your app is subject to U.S. export laws, regardless of where your legal entity is based. If your app uses, accesses, contains, implements, or incorporates encryption, this is considered an export of encryption software, which means your app is subject to U.S. export compliance requirements, as well as the import compliance requirements of the countries where you distribute your app.

Every time you submit a new version of your app, App Store Connect asks you questions to guide you through a compliance review. You can bypass these questions and streamline the submission process by providing the required information in your app's Information Property List file.

For more information about export compliance, read Export compliance overview.

4.1.3. Cryptography Complying with Encryption Export Regulations Declare Your App's Use of Encryption

Add the ITSAppUsesNonExemptEncryption key to your app's Info.plist file with a Boolean value that indicates whether your app uses encryption. Set the value to NO if your app—including any third-party libraries it links against—doesn't use encryption, or if it only uses forms of encryption that are exempt from export compliance documentation requirements. Otherwise, set it to YES.

Typically, the use of encryption that's built into the operating system—for example, when your app makes HTTPS connections using URLSession—is exempt from export documentation upload requirements, whereas the use of proprietary encryption is not. To determine whether your use of encryption is considered exempt, see Determine your export compliance requirements.

4.1.4. Cryptography Complying with Encryption Export Regulations Provide Compliance Documentation

If your app requires export compliance documentation, upload the required items to App Store Connect, as described in Upload export compliance documentation. After successfully reviewing the documents, Apple provides you with a code. Add this string as the value for the ITSEncryptionExportComplianceCode key in your app's Info.plist file.

4.1.5. Cryptography Certificate, Key, and Trust Services Overview

The certificate, key, and trust services API is a collection of functions and data structures that you use to conduct secure and authenticated data transactions. Specifically, you use this API to manage and use:

Certificates and identities. A certificate is a collection of data that identifies its owner in a tamper-evident way. When you use a certificate to distribute a public key, a receiver can be confident of its origin. You can also package a certificate together with its corresponding private key in an identity object that you keep secret.

Policies and trust services. When you receive a certificate, before you can use the embedded public key, you have to answer the question, "Can I trust this certificate?" You conduct an evaluation of trust according to a set of criteria, or a trust policy.

Cryptographic keys. After you have a key whose origin you trust, you can begin to conduct cryptographic operations, such as encryption or data signing and verification. These operations in turn typically serve a larger purpose, such as authenticating a user, transmitting data securely, or verifying that a block of data is unaltered since being sealed with a signature.

4.1.6. Cryptography Cryptographic Message Syntax Services Overview

When you want to exchange data securely using the Multipurpose Internet Mail Extensions (MIME) protocol, you use the version of the protocol known as S/MIME defined in RFC 3851. This allows you to, among other things, ensure data integrity through digital signatures and data confidentiality through encryption. S/MIME in turn relies on the Cryptographic Message Syntax (CMS) protocol defined in RFC 3852 to carry out these operations.

Cryptographic message syntax services provides encoder objects that perform encryption using the CMS protocol's enveloped-data content type and sign using the signed-data content type. When a message is both signed and encrypted, the enveloped data content contains the signed data content. That is, the message is first signed and then the signed content is encrypted.

4.1.7. Cryptography Randomization Services Overview

Many security operations rely on randomization to avoid reproducibility. This is true for many complex cryptographic operations, such as key generation, but is also true for something as simple as generating a password string that can't be easily guessed. If the string's characters are truly random (and kept hidden), an attacker has no choice but to try every possible combination one at a time in a brute force attack. For sufficiently long strings, this becomes unfeasible.

But the strength of such a password depends on the quality of the randomization. True randomization is not possible in a deterministic system, such as one where software instructions from a bounded set are executed according to well-defined rules. But even ""good"" randomization (in a statistical sense) is difficult to produce under these conditions. If an attacker can infer patterns in insufficiently randomized data, your system becomes compromised. Use randomization services to generate a cryptographically secure set of random numbers.

Figure 1 Random number generation
Diagram showing random number generation.

4.1.8. Cryptography Security Transforms Overview

You use security transforms to assemble a chain of security-related operations that you apply to a stream of data in macOS.

4.1.9. Cryptography Security Transforms Overview

> You use an ASN.1 coder to encode and decode both DER and BER data streams based on templates that you supply, which in turn are based upon ASN.1 specifications. You must import this API explicitly:
>
> import Security.SecAsn1Coder
> import Security.SecAsn1Templates

4.1.10. Legacy Interfaces Secure Transport Overview

> The Security.SecureTransport API gives you access to Apple's implementation of Secure Sockets Layer version 3.0 (SSLv3), Transport Layer Security (TLS) versions 1.0 through 1.2, and Datagram Transport Layer Security (DTLS) version 1.0.
>
> This API imposes no transport layer dependencies. You can use it with BSD Sockets and other protocols. To use this API, you provide callback functions to perform I/O on the underlying network connections. You are also responsible for setting up raw network connections. You pass in an opaque reference to the underlying (connected) entity at the start of an SSL session in the form of an SSLConnectionRef object.

5. US National Institute of Standards and Technology (NIST)

5.1. NIST Special Publication 800-163 Revision 1

Link: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-163r1.pdf

5.1.1. 2.2 Organization-Specific Requirements Digital Signature

> Digital signatures applied to the app binaries, libraries, or packages.

5.2. NIST Special Publication 800-190

Link: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-190.pdf

5.2.1. 4.1.4 Embedded clear text secrets

> Secrets should be stored outside of images and provided dynamically at runtime as needed. Most orchestrators, such as Docker Swarm and Kubernetes, include native management of secrets. These orchestrators not only provide secure storage of secrets and ‘just in time’ injection to containers, but also make it much simpler to integrate secret management into the build and deployment processes. For example, an organization could use these tools to securely provision the database connection string into a web application container. The orchestrator can ensure that only the web application container had access to this secret, that it is not persisted to disk, and that anytime the web app is deployed, the secret is provisioned into it.
>
> Organizations may also integrate their container deployments with existing enterprise secret management systems that are already in use for storing secrets in non-container environments. These tools typically provide APIs to retrieve secrets securely as containers are deployed, which eliminates the need to persist them within images.
>
> Regardless of the tool chosen, organizations should ensure that secrets are only provided to the specific containers that require them, based on a pre-defined and administrator-controlled setting, and that secrets are always encrypted at rest and in transit using Federal Information Processing Standard (FIPS) 140 approved cryptographic algorithms contained in validated cryptographic modules.

6. Open Web Application Security Project (OWASP)

6.1. Mobile Application Security Verification Standard (MASVS)

Link: https://github.com/OWASP/owasp-masvs/releases/tag/v1.4.2

6.1.1. 3.1 MSTG-CRYPTO-1

> The app does not rely on symmetric cryptography with hardcoded keys as a sole method of encryption.

6.1.2. 3.2 MSTG-CRYPTO-2

> The app uses proven implementations of cryptographic primitives.

6.1.3. 3.3 MSTG-CRYPTO-3

> The app uses cryptographic primitives that are appropriate for the particular use-case, configured with parameters that adhere to industry best practices.

6.1.4. 3.4 MSTG-CRYPTO-4

The app does not use cryptographic protocols or algorithms that are widely considered deprecated for security purposes.

6.1.5. 3.5 MSTG-CRYPTO-5

The app doesn't re-use the same cryptographic key for multiple purposes.

6.1.6. 3.6 MSTG-CRYPTO-6

All random values are generated using a sufficiently secure random number generator.

6.1.7. 5.2 MSTG-NETWORK-2

The TLS settings are in line with current best practices, or as close as possible if the mobile operating system does not support the recommended standards.

6.1.8. 7.1 MSTG-CODE-1

The app is signed and provisioned with a valid certificate, of which the private key is properly protected.

6.2. Application Security Verification Standard 4.0.3 (ASVS)

Link: https://raw.githubusercontent.com/OWASP/ASVS/v4.0.3/4.0/OWASP%20Application%20Security%20Verification%20Standard%204.0.3-en.pdf

6.2.1. V1.6 Cryptographic Architecture

1.6.1 Verify that there is an explicit policy for management of cryptographic keys and that a cryptographic key lifecycle follows a key management standard such as NIST SP 800-57.

1.6.2 Verify that consumers of cryptographic services protect key material and other secrets by using key vaults or API based alternatives.

6.2.2. V2.8 One Time Verifier

2.8.3 Verify that approved cryptographic algorithms are used in the generation, seeding, and verification of OTPs.

6.2.3. V2.9 Cryptographic Verifier

2.9.2 Verify that the challenge nonce is at least 64 bits in length, and statistically unique or unique over the lifetime of the cryptographic device.

2.9.3 Verify that approved cryptographic algorithms are used in the generation, seeding, and verification.

6.2.4. V3.2 Session Binding

3.2.4 Verify that session tokens are generated using approved cryptographic algorithms.

6.2.5. V6.2 Algorithms

6.2.2 Verify that industry proven or government approved cryptographic algorithms, modes, and libraries are used, instead of custom coded cryptography.

6.2.3 Verify that encryption initialization vector, cipher configuration, and block modes are configured securely using the latest advice.

6.2.4 Verify that random number, encryption or hashing algorithms, key lengths, rounds, ciphers or modes, can be reconfigured, upgraded, or swapped at any time, to protect against cryptographic breaks.

6.2.5 Verify that known insecure block modes (i.e. ECB, etc.), padding modes (i.e. PKCS#1 v1.5, etc.), ciphers with small block sizes (i.e. Triple-DES, Blowfish, etc.), and weak hashing algorithms (i.e. MD5, SHA1, etc.) are not used unless required for backwards compatibility.

6.2.6 Verify that nonces, initialization vectors, and other single use numbers must not be used more than once with a given encryption key. The method of generation must be appropriate for the algorithm being used.

6.2.7 Verify that encrypted data is authenticated via signatures, authenticated cipher modes, or HMAC to ensure that ciphertext is not altered by an unauthorized party.

6.2.8 Verify that all cryptographic operations are constant-time, with no 'short-circuit' operations in comparisons, calculations, or returns, to avoid leaking information.

6.2.6. V6.3 Random Values

6.3.1 Verify that all random numbers, random file names, random GUIDs, and random strings are generated using the cryptographic module's approved cryptographically secure random number generator when these random values are intended to be not guessable by an attacker.

6.3.2 Verify that random GUIDs are created using the GUID v4 algorithm, and a Cryptographically-secure Pseudo-random Number Generator (CSPRNG).
GUIDs created using other pseudo-random number generators may be
predictable.

6.3.3 Verify that random numbers are created with proper entropy even when the application is under heavy load, or that the application degrades gracefully in such circumstances.

6.2.7. V6.4 Secret Management

6.4.2 Verify that key material is not exposed to the application but instead uses an isolated security module like a vault for cryptographic operations.

6.2.8. V8.3 Sensitive Private Data

8.3.7 Verify that sensitive or private information that is required to be encrypted, is encrypted using approved algorithms that provide both confidentiality and integrity.

6.2.9. V9.1 Client Communication Security

9.1.1 Verify that TLS is used for all client connectivity, and does not fall back to insecure or unencrypted communications.

9.1.2 Verify using up to date TLS testing tools that only strong cipher suites are enabled, with the strongest cipher suites set as preferred.

9.1.3 Verify that only the latest recommended versions of the TLS protocol are enabled, such as TLS 1.2 and TLS 1.3. The latest version of the TLS protocol should be the preferred option.

6.2.10. V9.2 Server Communication Security

9.2.1 Verify that connections to and from the server use trusted TLS certificates. Where internally generated or self-signed certificates are used, the server must be configured to only trust specific internal CAs and specific self-signed certificates. All others should be rejected.

9.2.4 Verify that proper certification revocation, such as Online Certificate Status Protocol (OCSP) Stapling, is enabled and configured.

7. ioXt Alliance

7.1. Mobile Application Profile

Link: https://static1.squarespace.com/static/5c6dbac1f8135a29c7fbb621/t/604aa3fa668a8e3b50630433/1615504379349/Mobile_Application_Profile.pdf

7.1.1. 4.3. Proven Cryptography PC1

Standard cryptography

7.1.2. 4.3. Proven Cryptography PC2

Independently reviewed protocol, implementation, or open standard

7.1.3. 4.3. Proven Cryptography PC103

Store cryptographic private keys in the OS KeyStore.

7.1.4. 4.5. Verified Software VS3

Proven Cryptography.

7.1.5. 4.7. Secured Interfaces SI101

Encrypt all network traffic, using verified TLS where possible.

7.1.6. 4.7. Secured Interfaces SI104

Enforce x509 certificate pinning for primary services.