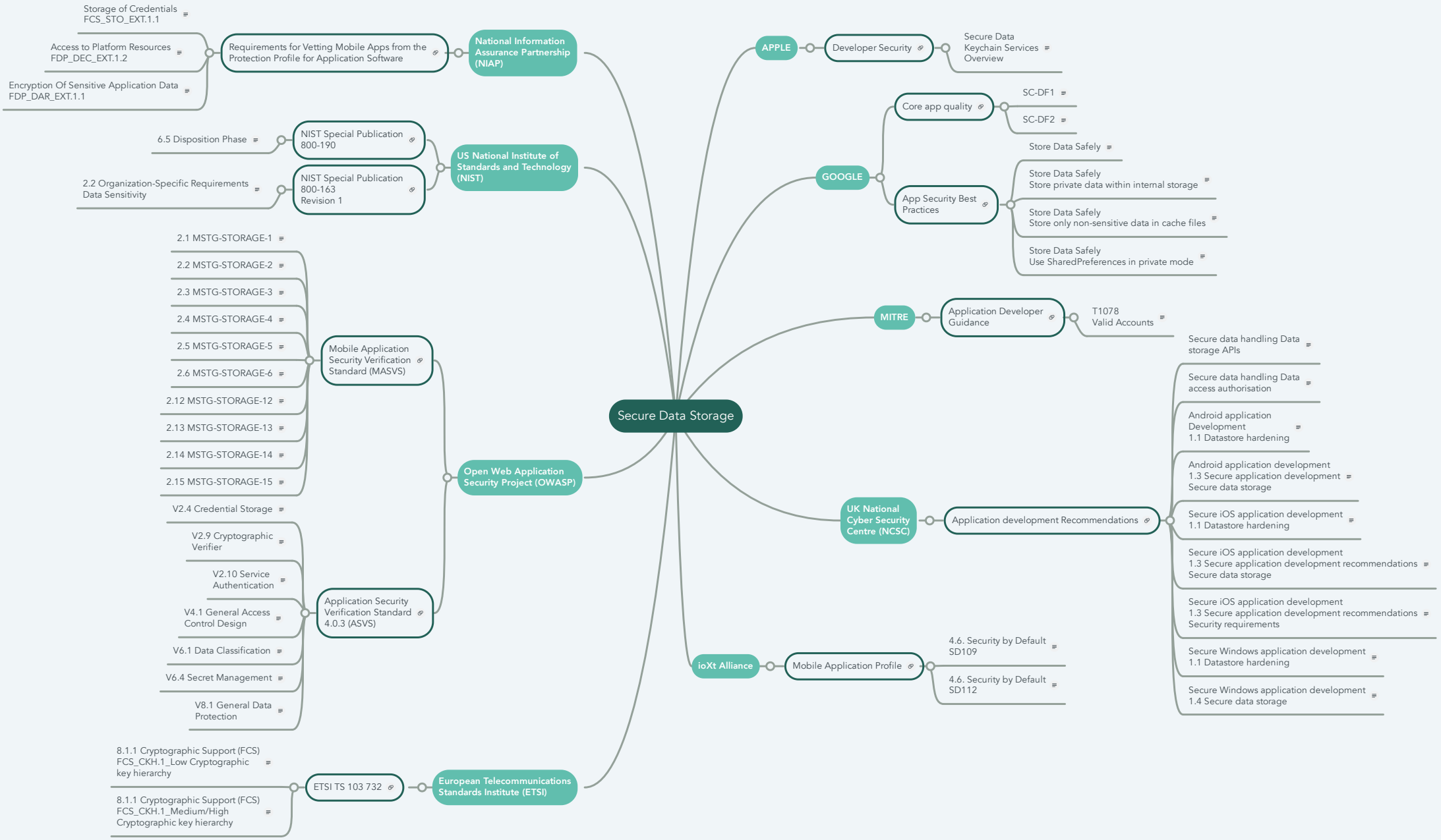
Secure Data Storage
National Information Assurance Partnership (NIAP)
Requirements for Vetting Mobile Apps from the Protection Profile for Application Software
Storage of Credentials FCS_STO_EXT.1.1
Access to Platform Resources FDP_DEC_EXT.1.2
Encryption Of Sensitive Application Data FDP_DAR_EXT.1.1
US National Institute of Standards and Technology (NIST)
NIST Special Publication 800-190
6.5 Disposition Phase
NIST Special Publication 800-163 Revision 1
2.2 Organization-Specific Requirements Data Sensitivity
Open Web Application Security Project (OWASP)
Mobile Application Security Verification Standard (MASVS)
2.1 MSTG-STORAGE-1
2.2 MSTG-STORAGE-2
2.3 MSTG-STORAGE-3
2.4 MSTG-STORAGE-4
2.5 MSTG-STORAGE-5
2.6 MSTG-STORAGE-6
2.12 MSTG-STORAGE-12
2.13 MSTG-STORAGE-13
2.14 MSTG-STORAGE-14
2.15 MSTG-STORAGE-15
Application Security Verification Standard 4.0.3 (ASVS)
V2.4 Credential Storage
V2.9 Cryptographic Verifier
V2.10 Service Authentication
V4.1 General Access Control Design
V6.1 Data Classification
V6.4 Secret Management
V8.1 General Data Protection
European Telecommunications Standards Institute (ETSI)
ETSI TS 103 732
8.1.1 Cryptographic Support (FCS) FCS_CKH.1_Low Cryptographic key hierarchy
8.1.1 Cryptographic Support (FCS) FCS_CKH.1_Medium/High Cryptographic key hierarchy
APPLE
Developer Security
Secure Data Keychain Services Overview
GOOGLE
Core app quality
SC-DF1
SC-DF2
App Security Best Practices
Store Data Safely
Store Data Safely Store private data within internal storage
Store Data Safely Store only non-sensitive data in cache files
Store Data Safely Use SharedPreferences in private mode
MITRE
Application Developer Guidance
T1078 Valid Accounts
UK National Cyber Security Centre (NCSC)
Application development Recommendations
Secure data handling Data storage APIs
Secure data handling Data access authorisation
Android application Development 1.1 Datastore hardening
Android application development 1.3 Secure application development Secure data storage
Secure iOS application development 1.1 Datastore hardening
Secure iOS application development 1.3 Secure application development recommendations Secure data storage
Secure iOS application development 1.3 Secure application development recommendations Security requirements
Secure Windows application development 1.1 Datastore hardening
Secure Windows application development 1.4 Secure data storage
ioXt Alliance
Mobile Application Profile
4.6. Security by Default SD109
4.6. Security by Default SD112

Secure Data Storage

1. APPLE

1.1. Developer Security

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

1.1.1. Secure Data Keychain Services Overview

> Computer users often have small secrets that they need to store securely. For example, most people manage numerous online accounts. Remembering a complex, unique password for each is impossible, but writing them down is both insecure and tedious. Users typically respond to this situation by recycling simple passwords across many accounts, which is also insecure.
>
> The keychain services API helps you solve this problem by giving your app a mechanism to store small bits of user data in an encrypted database called a keychain. When you securely remember the password for them, you free the user to choose a complicated one.
>
> The keychain is not limited to passwords, as shown in Figure 1. You can store other secrets that the user explicitly cares about, such as credit card information or even short notes. You can also store items that the user needs but may not be aware of. For example, the cryptographic keys and certificates that you manage with Certificate, Key, and Trust Services enable the user to engage in secure communications and to establish trust with other users and devices. You use the keychain to store these items as well.

2. GOOGLE

2.1. Core app quality

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

2.1.1. SC-DF1

> All sensitive data is stored in the app's internal storage.

2.1.2. SC-DF2

> No personal or sensitive user data is logged to the system log or an app-specific log.

2.2. App Security Best Practices

Link: https://developer.android.com/topic/security/best-practices

2.2.1. Store Data Safely

> Although your app might require access to sensitive user information, your users will grant your app access to their data only if they trust that you'll safeguard it properly.

2.2.2. Store Data Safely Store private data within internal storage

> Store all private user data within the device's internal storage, which is sandboxed per app. Your app doesn't need to request permission to view these files, and other apps cannot access the files. As an added security measure, when the user uninstalls an app, the device deletes all files that the app saved within internal storage.

2.2.3. Store Data Safely Store only non-sensitive data in cache files

> To provide quicker access to non-sensitive app data, store it in the device's cache. For caches larger than 1 MB in size, use getExternalCacheDir(); otherwise, use getCacheDir(). Each method provides you with the File object that contains your app's cached data.

2.2.4. Store Data Safely Use SharedPreferences in private mode

> When using getSharedPreferences() to create or access your app's SharedPreferences objects, use MODE_PRIVATE. That way, only your app can access the information within the shared preferences file.
>
> If you want to share data across apps, don't use SharedPreferences objects. Instead, you should follow the necessary steps to share data securely across apps.

3. MITRE

3.1. Application Developer Guidance

Link: https://attack.mitre.org/mitigations/M1013/

3.1.1. T1078 Valid Accounts

> Ensure that applications do not store sensitive data or credentials insecurely. (e.g. plaintext credentials in code, published credentials in repositories, or credentials in public cloud storage).

4. UK National Cyber Security Centre (NCSC)

4.1. Application development Recommendations

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

4.1.1. Secure data handling Data storage APIs

> You should not store sensitive information on a device when it's not required. If it must be stored on a device, make use of any native data storage protection APIs available on the platform.
> Make a model of the data flow in and out of the device, taking into consideration realistic threats that the system (and its users) may encounter.
> Most platforms provide documented APIs that allow data to be stored with different levels of protection.
> Always encrypt sensitive information when stored, protected by an authentication mechanism such as a passcode or platform-specific equivalent. When the sensitive data is no longer required on the device, it should be securely removed.
> Ensure the applications allows administrators to delete sensitive data from devices if they are compromised or lost.

4.1.2. Secure data handling Data access authorisation

> Store sensitive information securely, and hide it from the user until they have been authenticated (and authorised) to view it.
> Perform user authentication using the native platform mechanisms, with each account being linked to an individual.
> Where practical, manage user accounts centrally.
> When the application has lost focus (or been backgrounded for a short amount of time), the authentication process should be repeated to ensure the identity or permissions of the current user have not changed.

4.1.3. Android application Development 1.1 Datastore hardening

Android, by default, provides each application on a device with access to a private directory to store its files. This protection is implemented using Linux user and group permissions. The security model is further enforced by applying Security-Enhanced Linux mandatory access control policies and leveraging a seccomp system call filter.

Android, as of version 7.0, on devices with file-based encryption (FBE), provides two storage locations on devices with FBE and Direct Boot. On FBE-enabled devices, developers should only store sensitive data in the Credential Encrypted (CE) storage.

Applications are able to access other areas of the device, such as contacts and SMS, by requesting permission from the user at runtime. The user can choose to permit the application access to areas such as the device's calendar and phonebook, as well as features such as making phone calls or reading the current location. Once permitted, the application may use these features without further interaction from the user.

Despite protection offered by Android's sandboxing, it remains the responsibility of the application to store its data securely and to not undermine any protections that are in place by (for instance):

writing data to publicly readable locations such as the external storage
handling intents that can be called by any other application on the same device
creating files with world readable/writable permissions
Remember that a process running on the device with sufficient permissions, will always be able to read and write any data in any application's sandbox. We strongly recommended that applications holding sensitive data should build upon the sandbox with more secure functionality by (for example) leveraging the hardware-backed KeyStore.

Ultimately, it is not possible to guarantee the security of data on a device. You should assume that if a user continues to use a device after it has been compromised, the malware will be able to access the data. Android provides an API called SafetyNet for assessing the health and safety of the device. This API examines both hardware and software information about the device, to help determine if it has been tampered with or otherwise modified. We recommend that application developers use the API, then send the signed SafetyNet API results to their own servers to be validated, rather than on the device. SafetyNet should be used as a means to gain confidence about the integrity of the device, but it is not guaranteed to detect a compromise.

4.1.4. Android application development 1.3 Secure application development Secure data storage

In order to store sensitive data in a secure manner, Android applications should conform to the following:

Applications should minimise the amount of data stored on the device. When needed, data should be retrieved from the server over a secure connection, and erased when it is no longer required.
Sensitive information, if required, should only be stored in the hardware-backed KeyStore.
The device’s external storage (for example the SD card) should not be used by the application to store sensitive data.

4.1.5. Secure iOS application development 1.1 Datastore hardening

By default, third party App Store applications on iOS will be able to ask for access the users’ Calendars, Contacts, Camera, Location, Photos, and Social Networking accounts. On iOS these accesses are prompted on first use in each application, such that the user can accept or decline the permission. You can configure Restrictions settings on the device to prevent this functionality being used. Within an organisation, this is typically configured and deployed using an MDM-based solution. More information about MDM managed devices can be found within NCSC’s end user device guidance.

Nevertheless, there remains the possibility that the user could accept these access permissions and the application could access data in these stores. If there's a risk of an untrusted app accessing this data, then you should not store sensitive information within these datastores. A third party application may be able to store this information more securely than the default stores.

As the potential exists that the device may be compromised, on-device encryption routines cannot be solely relied on to protect sensitive information. Sensitive information should not be stored on a device for longer than it is required. Where sensitive information is stored on the device, even if temporarily, the following steps should be taken:

Sensitive credentials should be sufficiently encrypted before being stored within the keychain using the appropriate keychain class described in section 1.3 below.
The appropriate data protection class should still be used.
Sensitive data stored by the application should be marked with the 'do not backup' attribute to ensure that specified files are not included within an iTunes or iCloud backup.

4.1.6. Secure iOS application development 1.3 Secure application development recommendations Secure data storage

In order to store sensitive data in a secure manner, iOS applications should conform to the following:

Applications should use the iOS Keychain APIs to store credentials.
Applications should use the iOS Data Protection API to store sensitive file system data.
Applications should store as much data as possible using data protection classes A and B.
Private keys should be marked as non-migratory.
Where a credential is required to authenticate to a remote service that provides access to sensitive data, applications must store this credential in Class A or C keychain protection classes, or prompt for the credential on application launch.
The kSecAttrAccessibleWhenPasscodeSetThisDeviceOnly attribute could also be used to further harden local credential storage to prevent any synchronisation with the iCloud keychain. This acts the same as kSecAttrAccessibleWhenUnlocked. However, it is only available when devices have a passcode set and is protected via hardware-backed storage mechanisms.
Developers should be careful when storing information in the cloud. iCloud or other Internet-based storage solutions should not be used to store sensitive information (e.g. credentials). The application must work as expected if iCloud is disabled on the device.
Certain keychain APIs can be used to further constrain specific keychain items. To only allow access for Touch ID only, use kSecAccessControlTouchIDCurrentSet. A similar effect can be achieved using the secure enclave hardware-based key manager (kSecAttrTokenIDSecureEnclave). These attributes should be investigated to ensure the appropriate level of protection is implemented during development of a feature. More information about the secure enclave can be found within the API documentation.
On iOS version 9 and later, it is also possible to prevent a physical attacker from enrolling their own fingerprint on the device. This can be performed by reading the evaluatedPolicyDomainState variable to determine if TouchID enrolment changes have occurred since last usage.
We recommend that the application performs a wipe of its keychain data on first install (not upgrade). This can prevent keychain data being reused if the device is, for example, sold at a later stage and a full device wipe has not been performed.
Mask off all sensitive data on screen when the application receives notifications that will enter the background state using applicationWillResignActive and applicationDidEnterBackground. This is to ensure that the screenshot taken of the application does not contain sensitive information.
Network communication can also be cached within certain databases within the application sandbox. Ensure any network-level caching is not performed when sensitive data is being retrieved from the server side. Certain iOS APIs perform caching of network traffic (including plaintext data sent via HTTPS) on the device. If an attacker is able to gain access to the contents of the sandbox, they may be able to recover this data. Therefore, caching related APIs should be reviewed to ensure that sensitive data is not stored. More information can be found within Apple’s cache policy documentation.

4.1.7. Secure iOS application development 1.3 Secure application development recommendations Security requirements

The following list includes some other behaviours which can increase the overall security of an application.

Applications should store as much data as possible in data protection classes A and B (as described at the top of this page).
Applications should sanitise in-memory buffers after use, where possible, if the data is no longer required for operation (for example a temporary password or PIN buffer).
Applications should not upgrade the storage class of an existing file from Class D to a higher class. Instead, create a new file and copy the data across before deleting the original file. This ensures that the file is wrapped with a new key that may not be forensically recovered from Class D analysis.
Applications should minimise the amount of data stored on the device. Retrieve data from the server when needed, over a secure connection, and erase it when it is no longer required.
Applications that require authentication when launched should also request this authentication credential when returning back into the foreground after previously being backgrounded by a user, allowing for a small grace period.

4.1.8. Secure Windows application development 1.1 Datastore hardening

Universal Windows Platform (UWP) applications run in a container, meaning data storage is achieved in a sandboxed environment with its own file system and registry. Additionally, contained applications have restricted access to files on the host system or to data stored by other applications. Application developers do not need to implement anything to take advantage of this secure storage capability.

In cases where it is necessary to share data, UWP provides secure functionality for the following:

reading specific files on the host operating system as selected by the user
data sharing between applications
data storage for the same application across multiple Windows users
Developers should utilise the platform’s native features for those purposes.

Note that while UWP implements the principle of least privilege and ensures its applications cannot access external resources directly, applications with administrator privileges will still be able to read and write data to a UWP application.

4.1.9. Secure Windows application development 1.4 Secure data storage

Data-at-rest should be protected with use of the encryption and hashing APIs provided by UWP:

The SymmetricKeyAlgorithmProvider and AsymmetricKeyAlgorithmProvider classes should be used to implement encryption.
The CryptographicEngine class provides encryption, decryption, digital signing and signature verification capabilities.
The Security.Cryptography.DataProtection.DataProtectionProvider class should be used to encrypt and decrypt stored local data.
UWP provides a range of functionality with built-in support for use of a device's Trusted Platform Module (TPM), which protects against a range of attacks. Data protected by a TPM is very difficult for an attacker to access. The following TPM based functionality can be used to store data securely:

Platform Crypto Provider gives access to robust cryptography schemes, including those that are backed by the TPM. It can be used to securely store data on the device.
Windows Hello has integrated TPM support and can be used to authenticate and validate users for access to sensitive datastores.
Cryptographic keys and sensitive data should not be stored on the device unless they are stored using a TPM via a UWP feature such as those listed above.

When storing credentials, the Credential Locker feature should be used as it prevents other UWP applications from accessing them. Note that non-UWP applications and elevated users are able to access credentials within the Credential Locker, so for increased protection they could be encrypted before being stored. Credential Locker documentation details the following best practices for its use:

only use the credential locker for passwords and not for data blobs
never store credentials in plain-text using app data or roaming settings
only save passwords in the credential locker if the user has:
a) successfully signed in
b) opted to save passwords

5. ioXt Alliance

5.1. Mobile Application Profile

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

5.1.1. 4.6. Security by Default SD109

Store sensitive data only within the application container or system credential storage facilities.

5.1.2. 4.6. Security by Default SD112

> No sensitive data is leaked in the UI.

6. European Telecommunications Standards Institute (ETSI)

6.1. ETSI TS 103 732

Link: https://www.etsi.org/deliver/etsi_ts/103700_103799/103732/01.01.01_60/ts_103732v010101p.pdf

6.1.1. 8.1.1 Cryptographic Support (FCS) FCS_CKH.1_Low Cryptographic key hierarchy

> FCS_CKH.1.1/Low The TSF shall support a key hierarchy for the data encryption key(s) for Low user data assets.
> FCS_CKH.1.2/Low The TSF shall ensure that all keys in the key hierarchy are derived and/or generated according to [assignment: description of how each key in the hierarchy is derived and/or generated, with which key lengths and according to which standards] ensuring that the key hierarchy uses the DUK directly or indirectly in the derivation of the data encryption key(s) for Low user data assets.
> NOTE 2: The key derivation process requires that the data encryption key(s) for Low user data assets is derived directly from DUK or from another key which further derived from DUK. This ensures that Low user data assets can only be decrypted in the device which has the DUK. Example of Key derivation algorithm can be DRBG AES-256-CTR which meets NIST SP 800-90A [i.15].
> FCS_CKH.1.3/Low The TSF shall ensure that all keys in the key hierarchy and all data used in deriving the keys in the hierarchy are protected according to [assignment: rules].
> EXAMPLE 1: Examples of rules can be: keys and all data used in deriving the keys are stored in hardware based secure environment as defined in FPT_PHP.3.1, keys and all data used in deriving the keys are encrypted, or other rules defined by the TOE developer.

6.1.2. 8.1.1 Cryptographic Support (FCS) FCS_CKH.1_Medium/High Cryptographic key hierarchy

FCS_CKH.1.1/MediumHigh The TSF shall support a key hierarchy for the data encryption keys for Medium and High user data assets.
FCS_CKH.1.2/MediumHigh The TSF shall ensure that all keys in the key hierarchy are derived and/or generated according to [assignment: description of how each key in the hierarchy is derived and/or generated, with which key lengths and according to which standards] ensuring that the key hierarchy:
• uses the DUK directly or indirectly in the derivation of the data encryption keys for Medium and High user data assets; and
• uses the PIN, password, pattern or biometric template directly or indirectly in the derivation of the data encryption keys for Medium and High user data assets.
NOTE 3: The key derivation process requires that the data encryption key(s) for Medium/High user data assets is derived from a combination of the DUK and user authentication information. This ensures that Medium/High user data assets can only be decrypted in the device which has the DUK and after user is successfully authenticated. Example of Key derivation algorithm can be DRBG AES-256-CTR which meets NIST SP 800-90A [i.15].
FCS_CKH.1.3/MediumHigh The TSF shall ensure that all keys in the key hierarchy and all data used in deriving the keys in the hierarchy are protected according to [assignment: rules].
EXAMPLE 2: Examples of rules can be: keys and all data used in deriving the keys are stored in hardware based secure environment as defined in FPT_PHP.3.1, keys and all data used in deriving the keys are encrypted, or other rules defined by the TOE developer.

7. Open Web Application Security Project (OWASP)

7.1. Mobile Application Security Verification Standard (MASVS)

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

7.1.1. 2.1 MSTG-STORAGE-1

System credential storage facilities need to be used to store sensitive data, such as PII, user credentials or cryptographic keys.

7.1.2. 2.2 MSTG-STORAGE-2

No sensitive data should be stored outside of the app container or system credential storage facilities.

7.1.3. 2.3 MSTG-STORAGE-3

No sensitive data is written to application logs.

7.1.4. 2.4 MSTG-STORAGE-4

No sensitive data is shared with third parties unless it is a necessary part of the architecture.

7.1.5. 2.5 MSTG-STORAGE-5

The keyboard cache is disabled on text inputs that process sensitive data.

7.1.6. 2.6 MSTG-STORAGE-6

No sensitive data is exposed via IPC mechanisms.

7.1.7. 2.12 MSTG-STORAGE-12

The app educates the user about the types of personally identifiable information processed, as well as security best practices the user should follow in using the app.

7.1.8. 2.13 MSTG-STORAGE-13

No sensitive data should be stored locally on the mobile device. Instead, data should be retrieved from a remote endpoint when needed and only be kept in memory.

7.1.9. 2.14 MSTG-STORAGE-14

If sensitive data is still required to be stored locally, it should be encrypted using a key derived from hardware backed storage which requires authentication.

7.1.10. 2.15 MSTG-STORAGE-15

The app's local storage should be wiped after an excessive number of failed authentication attempts.

7.2. 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

7.2.1. V2.4 Credential Storage

2.4.1 Verify that passwords are stored in a form that is resistant to offline attacks. Passwords SHALL be salted and hashed using an approved one-way key derivation or password hashing function. Key derivation and password hashing functions take a password, a salt, and a cost factor as inputs when generating a password hash.

2.4.2 Verify that the salt is at least 32 bits in length and be chosen arbitrarily to minimize salt value collisions among stored hashes. For each credential, a unique salt value and the resulting hash SHALL be stored.

2.4.3 Verify that if PBKDF2 is used, the iteration count SHOULD be as large as verification server performance will allow, typically at least 100,000 iterations.

2.4.4 Verify that if bcrypt is used, the work factor SHOULD be as large as verification server performance will allow, with a minimum of 10.

2.4.5 Verify that an additional iteration of a key derivation function is performed, using a salt value that is secret and known only to the verifier. Generate the salt value using an approved random bit generator [SP 800-90Ar1] and provide at least the minimum security strength specified in the latest revision of SP 800-131A. The secret salt value SHALL be stored separately from the hashed passwords (e.g., in a specialized device like a hardware security module).

7.2.2. V2.9 Cryptographic Verifier

2.9.1 Verify that cryptographic keys used in verification are stored securely and protected against disclosure, such as using a Trusted Platform Module (TPM) or Hardware Security Module (HSM), or an OS service that can use this secure storage.

7.2.3. V2.10 Service Authentication

2.10.3 Verify that passwords are stored with sufficient protection to prevent offline recovery attacks, including local system access.

2.10.4 Verify passwords, integrations with databases and third-party systems, seeds and internal secrets, and API keys are managed securely and not included in the source code or stored within source code repositories. Such storage SHOULD resist offline attacks. The use of a secure software
key store (L1), hardware TPM, or an HSM (L3) is recommended for password storage.

7.2.4. V4.1 General Access Control Design

> 4.1.2 Verify that all user and data attributes and policy information used by access controls cannot be manipulated by end users unless specifically authorized.

7.2.5. V6.1 Data Classification

> 6.1.1 Verify that regulated private data is stored encrypted while at rest, such as Personally Identifiable Information (PII), sensitive personal information, or data assessed likely to be subject to EU's GDPR.
>
> 6.1.2 Verify that regulated health data is stored encrypted while at rest, such as medical records, medical device details, or de-anonymized research records.
>
> 6.1.3 Verify that regulated financial data is stored encrypted while at rest, such as financial accounts, defaults or credit history, tax records, pay history, beneficiaries, or de-anonymized market or research records.

7.2.6. V6.4 Secret Management

> 6.4.1 Verify that a secrets management solution such as a key vault is used to securely create, store, control access to and destroy secrets.

7.2.7. V8.1 General Data Protection

> 8.1.1 Verify the application protects sensitive data from being cached in server components such as load balancers and application caches.
>
> 8.1.2 Verify that all cached or temporary copies of sensitive data stored on the server are protected from unauthorized access or purged/invalidated after the authorized user accesses the sensitive data.
>
> 8.1.6 Verify that backups are stored securely to prevent data from being stolen or corrupted.

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

8.1. NIST Special Publication 800-190

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

8.1.1. 6.5 Disposition Phase

The ability for containers to be deployed and destroyed automatically based on the needs of an app allows for highly efficient systems but can also introduce some challenges for records retention, forensic, and event data requirements. Organizations should make sure that appropriate mechanisms are in place to satisfy their data retention policies. Examples of issues that should be addressed are how containers and images should be destroyed, what data should be extracted from a container before disposal and how that data extraction should be performed, how cryptographic keys used by a container should be revoked or deleted, etc.

Data stores and media that support the containerized environment should be included in any disposal plans developed by the organization.

8.2. NIST Special Publication 800-163 Revision 1

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

8.2.1. 2.2 Organization-Specific Requirements Data Sensitivity

The sensitivity of data collected, stored, or transmitted by the app.

9. National Information Assurance Partnership (NIAP)

9.1. 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

9.1.1. Storage of Credentials FCS_STO_EXT.1.1

The application shall [selection:
not store any credentials, invoke the functionality provided by the platform to securely store [assignment: list of credentials] , implement functionality to securely store [assignment: list of credentials]
] to non-volatile memory. Application Note: This requirement ensures that persistent credentials (secret keys, PKI private keys, or passwords) are stored securely.
The assurance activity implicitly restricts which selections can be made, on per-platform basis. For example, if a platform provides hardware-backed protection for credential storage, then the third selection cannot be indicated.
If implement functionality to securely store credentials is selected, then the following components must be included in the ST: FCS_COP.1(1). If other cryptographic operations are used to implement the secure storage of credentials, the corresponding requirements must be included in the ST.

9.1.2. Access to Platform Resources FDP_DEC_EXT.1.2

The application shall restrict its access to [selection:
no sensitive information repositories,
address book,
calendar,
call lists,
system logs,
[assignment: list of additional sensitive information repositories]
] .
Application Note: Sensitive information repositories are defined as those collections of sensitive data that could be expected to be shared among some applications, users, or user roles, but to which not all of these would ordinarily require access.

9.1.3. Encryption Of Sensitive Application Data FDP_DAR_EXT.1.1

The application shall [selection:
leverage platform-provided functionality to encrypt sensitive data,
implement functionality to encrypt sensitive data, not store any sensitive data] in non-volatile memory.
Application Note: If implement functionality to encrypt sensitive data is selected, then evaluation is required against the Application Software Protection Profile Extended Package: File Encryption.
Any file that may potentially contain sensitive data (to include temporary files) shall be protected. The only exception is if the user intentionally exports the sensitive data to non-protected files.