Evidence of Compliance to the ETSI TS 103 732-1/TS 103 732-2 and GSMA Requirements

Overview

The general purpose of this document is to provide a natural language translation of the Common Criteria requirements that can be more readily understood. The requirements here are a combination of Security Functional Requirements (SFRs) and Security Assurance Requirements (SARs).

SFRs can generally be considered testable requirements on the device, though this is not always the case. In cases where it is not explicitly testable, generally evidence of meeting the requirement is provided, such as source code or pointing to another evaluation which proves the requirement is met. SFRs are always focused on the device itself, either a software or hardware component of the device which supports a security requirement.

SARs can generally be considered documentation requirements and may be focused on the device, on the production of the device (manufacturing) or in-market support. There may be some device testing done where it is possible to prove a requirement with a test (instead of documentation), but the majority of SARs will not require testing (this may change over time).

Beyond the TS 103 732 requirements, there are additional areas that require documentation to support the security evaluation of the device. These are listed after the TS 103 732 requirements.

How to complete the questionnaire:

<table>
<tr><td rowspan="4">TS 103 732 SFR, SAR or GSMA requirement</td><td>Requirement description</td><td rowspan="2">Supported? (Y/N)</td></tr>
<tr><td>Evidence required for submission</td></tr>
<tr><td>SFR from TS 103 732 or GSMA to be filled out</td><td rowspan="2">Y or N</td></tr>
<tr><td>OEM response goes here, depending on evidence requirement either provide a description, documentation or a reference to supporting material to be included with submission of the completed questionnaire.</td></tr>
</table>

Several items will need to be submitted along with the completed questionnaire. These may include, but not limited to:

- Process, policy, procedure, architectural and design documentation
- Source code listings, samples, and repositories

- Configuration data such as kernel defconfigs, bootloader configuration, SELinux policies
- Assessment reports for items such as privileged applications, OTA update services
- Devices configured as production, as well as userdebug, and any custom tools required for loading software

When answering the questionnaire, the set of devices being considered should be limited to those released in the last 12 months or with an impending release. Where appropriate, general responses covering all devices may be provided; device-specific responses should not be needed unless requirement implementation differs significantly.

The ETSI TS 103 732-1 questionnaire sections are grouped as they are in the Protection Profile:

- Cryptographic Support
- User Data Protection
- Identification and Authentication
- Security Management
- Privacy
- Protection of the TSF
- Trusted Path/Channels
-

The ETSI TS 103 732-2 sections:

- Identification and Authentication

The ETSI TS 103 732-4 sections:

- Applications
- Application Risk

The ETSI TS 103 732-5 sections:

- Bootloader - User Data Protection
- Bootloader - Protection of the TSF
- Bootloader - Life Cycle
- Root of Trust - Protection of the TSF

The GSMA FS.56 sections:

- Cryptographic Support
- Identification and Authentication
- Privacy
- Protection of the TSF
- Live Cycle Requirements

Devices for Certification

The following tables are for specifying the devices that will be certified. Add rows as necessary for each separate device covered in the evaluation.

Evaluated Devices

Device	Operating System	OS Version	Kernel Version
Pixel 10	Android	16	6.6

Device	Model(s)	CPU Architecture	CPU
Pixel 10		ARM64	Tensor G5

Equivalent Devices

The devices here are claimed by the developer as equivalent to an evaluated device.

Claimed Device	Evaluated Device	Differences between devices
Pixel 10 Pro XL	Pixel 10	Screen size, battery capacity, storage/memory capacity, case materials
Pixel 10 Pro	Pixel 10	Screen size, battery capacity, storage/memory capacity, case materials
Pixel 10 Pro Fold	Pixel 10	Screen size, battery capacity, storage/memory capacity, case materials, form factor

TS 103 732-1 SFRs

Cryptographic Support

This section is focused on the cryptography that is used in the system.This includes both key lifecycles and the cryptographic algorithms that are in use.

The requirements for cryptographic algorithms are open as long as the algorithm is able to meet the requirements set by ISO for adding the algorithm to the ISO standard. Note that this does not mean that the algorithm shall be submitted to ISO for inclusion in their standards, only that it meets those requirements. This ensures that the algorithm has been publicly available and reviewed, and so is considered acceptable for use to meet the security claims of this evaluation.

<table>
<tr><td rowspan="4">FCS_RNG_
EXT.1</td><td>Devices shall provide at least one random number generator (RNG) that produces uniformly-distributed, unpredictable output.</td><td rowspan="2">Supported?
(Y/N)</td></tr>
<tr><td>For each RNG on the device, provide a description of the type (such as physical or software) and how the amount of entropy provided has been verified. Common examples of proof would be NIST 800-90B or AIS 31 analysis.</td></tr>
<tr><td>FCS_RNG_EXT.1.1 The TSF shall provide a [physical, deterministic] random number generator.

FCS_RNG_EXT.1.2 The TSF shall provide random numbers that meet [hardware noise sources that provide at least .8bits of entropy per bit of output in the AP and Titan M2, a SHA-256 HMAC_DRBG in the Titan M2 and AES CTR_DRBG in BoringSSL].</td><td rowspan="2">Y</td></tr>
<tr><td>The device uses several different RNGs to provide sufficient entropy during key generation.

In hardware, there are two separate physical sources. The Application Processor (AP) and the Titan M2 chip each provide independent physical noise sources to generate random output.

Normal operating and startup output from both of the physical sources have been checked using the NIST 800-90B entropy test tools to provide sufficient entropy to ensure that the output from the hardware DRBGs will provide at least 1 bit of entropy for every bit requested. The DRBGs both use 384 bits of input to generate the 256-bit output blocks.

The Titan M2 chip noise source output is used to seed a SHA-256 HMAC_DRBG for use inside the chip.</td></tr>
</table>

	The AP output is provided to the system via the /dev/hw_random (or similar) device. The Android DRBGs are able to use the output from /dev/hw_random for seeding. As this device is normally restricted to only kernel processes, Android provides an entropy daemon that uses the BoringSSL AES CTR_DRBG to provide a constant source of entropy to user-space services (mainly BoringSSL itself). The DRBG in the entropy daemon is reseeded every 200 requests (well below the NIST 800-90B requirements for reseeding an approved DRBG function). In Android itself, when a request for a new random string is made, the AES CTR_DRBG that is provided as part of the BoringSSL library (libcrypto) is called. This DRBG is seeded by calling the entropy daemon for 384 bits to produce a 256-bit key. Each DRBG has been verified according to NIST testing: SoC DRBG: A7071 Titan M2 DRBG: A6963 BoringSSL DRBG: A6838	

FCS_CKM.1 /Asymmetric	Devices shall generate asymmetric keys properly that can meet at least the equivalent of 128-bit symmetric encryption strength. The process for generating asymmetric keys on the device shall be explained. This will be specific to the algorithm(s) in use.	Supported? (Y/N)
	FCS_CKM.1.1/Asymmetric The TSF shall generate cryptographic keys in accordance with a specified cryptographic key generation algorithm [RSA and ECDSA] and specified cryptographic key sizes [*RSA keys of 2048, 3072, 4096, ECDSA keys of P-256, P-384, P-521*] that meet the following: [FIPS 186-5].	***Y***
	BoringSSL and the Titan M2 chip both support generating RSA and ECDSA keys compliant with NIST FIPS 186-4 that are equivalent (or greater) than 112-bit symmetric strength (RSA 2048 and ECC 256). BoringSSL supports generating keys that are of 128-bit strength and higher for both RSA and ECDSA. In both cases the DRBG from FCS_RNG_EXT.1 is used in the process. Titan M2 DRBG: A6963 BoringSSL DRBG: A6838	

FCS_CKM.1 /Symmetric	Devices shall generate symmetric keys of at least 128-bit length either by generating the key using a RNG or by a derivation function.	Supported? (Y/N)
	The process for generating the keys on the device shall be explained. For example, the key is generated by calling the RNG.	
	FCS_CKM.1.1/Symmetric The TSF shall generate cryptographic keys in accordance with a specified cryptographic key generation algorithm [using AES CTR_DRBG in BoringSSL and SHA-256 HMAC_DRBG in the Titan M2] and specified cryptographic key sizes [256-bits] ~~that meet the following: [**assignment**: *list of standards*~~].	***Y***
	BoringSSL and the Titan M2 chip both support generating symmetric (AES) keys compliant with NIST FIPS 197 that are equivalent (or greater) than 128-bit symmetric strength. In both cases the DRBG from FCS_RNG_EXT.1 is used in the process.	

FCS_COP.1 Note

All the algorithms listed under FCS_COP.1 may have multiple implementations throughout the system. For example, symmetric encryption is likely to be available in low level hardware (i.e. SoC), a hardware storage encryption engine, a hardware-backed key store (if support is provided), the SEE, the kernel and within the main OS itself. Not all algorithms may be available in all components, and different configurations may provide different options.

The expectation for this section is that each instance of an algorithm type be specified if it is key to providing security services for the device (i.e. if an algorithm is available for user-installed applications or is not used by the security components of the device itself, they do not need to be listed).

FCS_COP.1 /SigGen	Devices shall support an asymmetric algorithm that is used to verify the signature of update packages (both for the OS and applications). Multiple algorithms may be supported for different purposes.	Supported? (Y/N)
	Each supported algorithm shall be listed along with the key sizes supported. The listing should point to the standard used to implement the algorithm (for example FIPS 186-4 for RSA).	
	FCS_COP.1.1/SigGen The TSF shall perform [*CRYPTOGRAPHIC SIGNATURE SERVICES (GENERATION AND VERIFICATION)*] in accordance with a specified cryptographic algorithm [RSA keys 2048-bits or greater, ECDSA	***Y***

	using NIST curves P256, P384, P-521] that meet the following: [FIPS PUB 186-5].	
	FIPS 186-5 support for both RSA and ECDSA (though Titan M2 only supports RSA 2048 and ECDSA P256). Titan M2 DRBG: A6963 BoringSSL DRBG: A6838	

FCS_COP.1 /KeyEst	Devices shall support a key establishment algorithm for the encryption/decryption of user data. This shall list the algorithm used for user data encryption in transit, and algorithms used in other parts of the system.	Supported? (Y/N)
	Each supported algorithm shall be listed along with the key sizes supported. The listing should point to the standard used to implement the algorithm (for example FIPS 197 for AES).	
	FCS_COP.1.1/Symmetric The TSF shall perform [*CRYPTOGRAPHIC KEY ESTABLISHMENT*] in accordance with a specified cryptographic algorithm [Elliptic curve-based key establishment schemes] and cryptographic key sizes [256 bits] that meet the following: [NIST Special Publication 800-56A Revision 3, "Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography"].	***Y***
	To support TLS communications, BoringSSL supports ECDH key pair generation. BoringSSL DRBG: A6838	

FCS_COP.1 /Symmetric	Devices shall support a symmetric algorithm for the encryption/decryption of user data. This shall list the algorithm used for user data encryption, and algorithms used in other parts of the system.	Supported? (Y/N)
	Each supported algorithm shall be listed along with the key sizes supported. The listing should point to the standard used to implement the algorithm (for example FIPS 197 for AES).	
	FCS_COP.1.1/Symmetric The TSF shall perform [*SYMMETRIC ENCRYPTION AND DECRYPTION*] in accordance with a specified cryptographic algorithm [AES] and cryptographic key sizes [256-bits] that meet the following: [AES as defined in FIPS PUB 197 with CBC (SP800-38A), CCM (SP800-38C), Key Wrap	***Y***

	(SP800-38F), GCM (SP800-38D), XTS (SP800-38E) and GCM (SP800-38D) modes].	
	Multiple modules provide symmetric encryption using AES. These algorithms are all verified as part of the NIST algorithm testing. AP (SoC): <under test> Storage (ISE): A7008 BoringSSL: A6838 Titan M2: A6963 Trusty TEE: A6842 Wi-Fi chip: A4158, A4159	

FCS_COP.1 /Derivation	Devices shall support a key derivation function.	Supported? (Y/N)
	Each supported function shall be listed along with the key sizes supported. The listing should point to the standard used to implement the algorithm (for example NIST SP 800-108 for KBKDF or NIST SP 800-132 for PBKDF2).	
	FCS_COP.1.1/Derivation The TSF shall perform [*DERIVATION FUNCTION*] in accordance with a specified cryptographic algorithm [HMAC-SHA2-256 and scrypt, CMAC-AES] and cryptographic key sizes [128 bits, 256 bits] that meet the following: [NIST SP 800-108 (CMAC-AES, HMAC-SHA2-256) and no standard (scrypt)].	***Y***
	Multiple modules provide KDF algorithms using NIST SP 800-108. These algorithms are all verified as part of the NIST algorithm testing. AP (SoC): A6766 Lockscreen settings: A2168 Titan M2: A6963 Trusty TEE: A6842 To derive a key from user input (password/PIN/pattern), the user input (and salt) is processed using scrypt (which contains a	

	PBKDF2 operation, followed by a series of ROMix operations and then one final PBKDF2). This key is then combined with a hardware-fused key using a KDF function inside the TEE (the output key is not used on its own, only in combination with the hardware-fused key).	

FCS_COP.1 /Hash	Devices shall support a cryptographic hash algorithm.	Supported? (Y/N)
	Each supported algorithm shall be listed along with the key sizes supported. The listing should point to the standard used to implement the algorithm (for example FIPS 180-4 for SHA).	
	FCS_COP.1.1/Hash The TSF shall perform [*CRYPTOGRAPHIC HASHING*] in accordance with a specified cryptographic algorithm [SHA2-256, 384 and 512] and cryptographic key sizes [*NONE*] that meet the following: [FIPS 180-4].	***Y***
	Multiple modules provide hash algorithms using SHA2. These algorithms are all verified as part of the NIST algorithm testing. AP (SoC): <under test> Storage (ISE) (for self-test): A7009 BoringSSL: A6838 Titan M2: A6963 Trusty TEE: A6842	

FCS_COP.1 /KeyedHash	Devices shall support a message authentication code algorithm.	Supported? (Y/N)
	Each supported algorithm shall be listed along with the key sizes supported. The listing should point to the standard used to implement the algorithm (for example FIPS 198-1 for HMAC).	
	FCS_COP.1.1/KeyedHash The TSF shall perform [*KEYED-HASH MESSAGE AUTHENTICATION*] in accordance with a specified cryptographic algorithm [HMAC-SHA2-256, 384, 512] and cryptographic key sizes [256, 384 and 512 bits] that meet the following: [FIPS 198-1 and FIPS 180-4].	***Y***
	Multiple modules provide keyed hash algorithms using SHA2. These algorithms are all verified as part of the NIST algorithm testing.	

	AP (SoC): <under test> Storage (ISE) (for self-test): A7009 BoringSSL: A6838 Titan M2: A6963 Trusty TEE: A6842	

FCS_CKH_ EXT.1/Low	Devices shall provide encrypted storage that is not tied to the user credentials. This can require separate apps to explicitly implement the functionality, but it shall be available in the device.	Supported? (Y/N)
	Documentation shall provide: ● Description of how DE keys are generated/derived ○ Algorithms, key lengths used in the process ● Description of how DE keys are cryptographically tied to hardware-backed keystore	
	FCS_CKH_EXT.1.1/Low The TSF shall support a key hierarchy for data encryption keys to protect [*LOW USER DATA ASSETS*]. **FCS_CKH_EXT.1.2/Low** The TSF shall ensure that all keys in the key hierarchy are derived and/or generated according to [file encryption keys are generated using AES_CTR DRBG from BoringSSL and are encrypted using a key that is derived from the DUK] ensuring that the key hierarchy uses the DUK and [NO USER CREDENTIALS] directly or indirectly in the derivation of the data encryption key(s) for [*LOW USER DATA ASSETS*]. **FCS_CKH_EXT.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 [no other rules].	***Y***
	https://source.android.com/docs/security/features/encryption/file-based https://developer.android.com/training/articles/direct-boot.html Android provides a key hierarchy tied to the device root key that allows some data to be encrypted by accessed before the user has successfully authenticated (such as phone calls or alarms). This follows the credentialed encryption key hierarchy but uses a hardcoded password instead of the user's password (after that	

	all data stored for DE storage is handled the same, just using the separate key hierarchy to handle the encryption keys). Use of this requires the app to explicitly take advantage of the DE storage capability. See Figure 4 in the KMD for a diagram See Tables 3 & 4 in the KMD for information about the keys involved (using a hardcoded password for the user password)	

FCS_CKH_EXT.1/MediumHigh	Devices shall provide encrypted storage that is tied to the user credentials. By default all user data shall be decrypted after the user authenticates the first time (device boot). Additionally the device shall provide the functionality to keep user data encrypted once the device has been unlocked but the user is not active (the user logged into the device at start-up and then the device has since been screen locked and requires the user to provide credentials again). This additional functionally can require separate apps to explicitly implement the functionality, but it shall be available in the device.	Supported? (Y/N)
	Documentation shall provide: ● Description of how CE keys are generated/derived ○ Algorithms, key lengths used in the process ● Description of how CE keys are cryptographically tied to hardware-backed keystore ● Description of how CE keys are cryptographically tangled with the user's credentials	
	FCS_CKH_EXT.1.1/MediumHigh The TSF shall support a key hierarchy for data encryption keys to protect [*MEDIUM AND HIGH USER DATA ASSETS*]. **FCS_CKH_EXT.1.2/MediumHigh** The TSF shall ensure that all keys in the key hierarchy are derived and/or generated according to [file and encryption keys are generated using AES_CTR DRBG from BoringSSL and are encrypted using a key that is derived from a combination of the user's credentials and the DUK] ensuring that the key hierarchy uses the DUK and [THE USER CREDENTIALS] directly or indirectly in the derivation of the data encryption key(s) for [*MEDIUM AND HIGH USER DATA ASSETS*]. **FCS_CKH_EXT.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 [no other rules].	***Y***

	MEDIUM https://source.android.com/docs/security/features/encryption/file-based https://developer.android.com/training/articles/direct-boot.html Android provides a key hierarchy tied to both the user's password and the device root key that protects all data by default such that it is not available until after a successful authentication from the user. This is called Credentialed Encryption (CE). The CE key hierarchy binds the master encryption key to both the user's credential and the root key from the device. If an app does not do anything specific to store data, it will be stored as CE data. See Figure 3 in the KMD for a diagram See Tables 3 & 4 in the KMD for information about the keys involved (using a hardcoded password for the user password) **HIGH** Android provides the ability for an app to be aware of the lock state of the device. This can be obtained through APIs: https://developer.android.com/reference/android/security/keystore/KeyGenParameterSpec.Builder#setUnlockedDeviceRequired(boolean) https://developer.android.com/reference/android/security/keystore/KeyGenParameterSpec.Builder#setUserAuthenticationRequired(boolean) When combined with the app being able to individually encrypt its own files: https://developer.android.com/topic/security/data#java An app can be developed that can maintain that its data is only able to be decrypted for use when the user is active on the device (the device is considered unlocked). This is an app-specific capability, and so requires an app to be developed to utilize this capability.	

FCS_CKM.4	Devices shall clear plain-text keys when no longer needed.	

	Describe how keys are cleared from memory (both volatile and non-volatile). Note that some of these may not be applicable in all devices (for example there may be no non-volatile flash memory that is not wear-leveled), in which case it is sufficient to specify it is not applicable.	Supported? (Y/N)
	FCS_CKM.4.1 The TSF shall destroy **keys from the key hierarchy for Low, Medium, and High** cryptographic keys in accordance with a specified cryptographic key destruction method [~~assignment:~~ • for non-volatile EEPROM, by a single direct overwrite consisting of a random pattern, using the TSF's RNG, followed by a read-verify; • for non-volatile flash memory, that is not wear-levelled, by [a block erase that erases the reference to memory that stores data as well as the data itself]; • for non-volatile flash memory, that is wear-levelled, by [a block erase]; • for non-volatile memory other than EEPROM and flash, by a single direct overwrite with a random pattern that is changed before each write]; that meets the following: [no standards].	***Y***
	The TOE clears sensitive cryptographic material (plaintext keys, authentication data, and other security parameters) from memory when no longer needed. Public keys can remain in memory when the phone is locked, but all crypto-related private keys are evicted from memory upon device lock. No plaintext cryptographic material resides in the TOE's Flash as the TOE encrypts all keys stored in Flash. When performing a full wipe of protected data, the TOE cryptographically erases the protected data by clearing the Data-At-Rest DEK. Because the Android Keystore of the TOE resides within the user data partition, the TOE effectively cryptographically erases those keys when clearing the Data-At-Rest DEK. In turn, the TOE clears the Data-At-Rest DEK and Secure Key Storage SEK through a secure direct overwrite (BLKSECDISCARD ioctl) of the wear-leveled Flash memory containing the key followed by a read-verify. See the Tables in the KMD for information about keys and clearing	

User Data Protection

FDP_ACC.1/APP_Update **FDP_ACF.1/APP_Update** **FDP_UPF_EXT.1/APP_Update**	All applications delivered via the app store (distribution platform) shall provide a means for applications to be updated. Document the following information about the update process: ● Frequency of automatic checking with the app store ● Method for verifying new updates are available (e.g. versioning such that older versions cannot be installed as an update) ● Method for verifying validity of the package (i.e. signature checks)	Supported? (Y/N)
	FDP_ACC.1.1/APP_Update The TSF shall enforce the [*APP_UPDATE POLICY*] on [*SUBJECTS: THE TSF, OBJECTS: APP, APP_UPDATE_PACKAGE, OPERATIONS: UPDATE_APP*]. **FDP_ACF.1.1/APP_Update** The TSF shall enforce the [*APP_UPDATE POLICY*] to objects based on the following: [*SUBJECTS: THE TSF, OBJECTS[ATTRIBUTES]: APP[VERSION_ID, SIGNATURE], APP_UPDATE_PACKAGE[VERSION_ID, SIGNATURE, PACKAGE_SOURCE]*]. **FDP_ACF.1.2/APP_Update** The TSF shall enforce the following rules to determine if an operation among controlled subjects and controlled objects is allowed: [● *THE TSF SHALL ALLOW THE TSF TO UPDATE_APP WITH AN APP_UPDATE_PACKAGE ONLY IF:* ○ *THE TSF SUCCESSFULLY VERIFIES THE SIGNATURE OF THE APP_UPDATE_PACKAGE AND THE SIGNATURE IS FROM THE SAME APP OR APP DEVELOPER; AND* ○ *[*the version ID of the App_Update_Package is not lower than the version ID of the installed App*];* ● *THE TSF SHALL UPDATE_APP IN AS AN ATOMIC UPDATE FUNCTION*]. **FDP_ACF.1.3/APP_Update** The TSF shall explicitly authorize access of subjects to objects based on the following additional rules: [no other rules]. **FDP_ACF.1.4/APP_Update** The TSF shall explicitly deny access of subjects to objects based on the following additional rules: [*THE TSF SHALL NOT ALLOW ANY TSF-MEDIATED ACTIONS RELATED TO THE UPDATE_APP OPERATION OR ACCESS TO THE APP DURING ITS UPDATING*].	***Y***

	FDP_UPF_EXT.1.1/APP_Update The TSF shall be able to check for a [APP] update package every [once a day`1].	
	The Play Store checks for app updates on a daily basis. Based on the user settings, the apps will be automatically updated when the device is not in use or the user will be notified that updates are available. The user can force a check for updates (or when notified start the update immediately). When an app update is available, it will only be installed if the signature is from the same app developer as the currently installed version and that the version ID is not lower than the currently installed version of the app. When an app is being updated, if the app is currently running, the instance will be closed during the update process. If the update fails for any reason, the installation process will roll back to the version that existed at the time of the download.	

FDP_ACC.2/SSW_Update **FDP_ACF.1/SSW_Update** **FDP_UPF_EXT.1/SSW_Update**	The device shall support system software updates (i.e. OTA updates) and secure how they are downloaded and installed.	Supported? (Y/N)
	Document the following information about the update process: • Frequency of automatic checking with the update location • Method for verifying new updates are available (e.g. versioning such that older versions cannot be installed as an update) • Method for verifying validity of the package (i.e. signature checks) • How security is maintained during the update process (i.e. that the system cannot be circumvented during the update process) ***NOTE: The yellow highlight is a modification from GSMA FS.56.***	
	FDP_ACC.2.1/SSW_Update The TSF shall enforce the [*SSW_UPDATE POLICY*] on [*SUBJECTS: THE TSF, OBJECTS: THE SSW, SSW_UPDATE_PACKAGE, OPERATIONS: UPDATE_SSW*]. **FDP_ACC.2.2/SSW_Update** The TSF shall ensure that all operations between any subject controlled by the TSF and any object controlled by the TSF are covered by an access control SFP. **FDP_ACF.1.1/SSW_Update** The TSF shall enforce the [*SSW_UPDATE POLICY*] to objects based on the following:	***Y***

	[*SUBJECTS: THE TSF, OBJECTS[ATTRIBUTES]: SSW[VERSION_ID, SIGNATURE], SSW_UPDATE_PACKAGE[VERSION_ID, SIGNATURE, PACKAGE_SOURCE]*]. **FDP_ACF.1.2/SSW_Update** The TSF shall enforce the following rules to determine if an operation among controlled subjects and controlled objects is allowed: [● *THE TSF IS ALLOWED TO PERFORM THE UPDATE_SSW OPERATION IF THE FOLLOWING CONDITIONS HOLD:* ○ *[*the SSW_Update_Package[version ID] is not lower than the SSW[version ID]*]* ○ *THE SSW_UPDATE_PACKAGE[SIGNATURE] IS VERIFIED BY A DIGITAL SIGNATURE FROM THE TOE MANUFACTURER STORED ON THE DEVICE* ○ *THE SIGNATURE CHECK AND THE UPDATE_SSW ARE PERFORMED AS AN ATOMIC UPDATE FUNCTION***]**. **FDP_ACF.1.3/SSW_Update** The TSF shall explicitly authorize access of subjects to objects based on the following additional rules: [no other rules]. **FDP_ACF.1.4/SSW_Update** The TSF shall explicitly deny access of subjects to objects based on the following additional rules: [*THE TSF SHALL NOT ALLOW ANY TSF-MEDIATED ACTIONS RELATED TO THE UPDATE_SSW FUNCTION DURING ITS UPDATING*]. **FDP_UPF_EXT.1.1/SSW_Update** The TSF shall be able to check for a [SYSTEM SOFTWARE] update package every once per day] and provide a notification to the user when an update is available.	
	The device verifies all updates using a public key chaining to the root public key. The SHA-256 hash of the root public key is fused into the SoC during manufacturing. Once the update has been downloaded the version and signature are checked to ensure the version is not older than the current version and that the signature is valid (verified to the hardware hash). https://source.android.com/docs/core/ota/ab If the checks complete successfully the update process begins immediately where the update will be installed on the slot	

	(partition) that is not currently in use (i.e. the one that was not used to boot the device at this time). Once the update has been completely written to the slot, the update process will switch the active slot to the one just updated and restart the device (with user permission). If the reboot is not successful the system will automatically switch back to the previous slot and restart again, returning the device to the previous state. If the checks, write process or reboot fail at any time during this sequence, the system will remain/revert to the OS version that existed prior to the update being downloaded, such that the update process happens completely or not at all. The device checks for updates on a daily basis (once every 24 hours) to the update server. When an update is found the user will be notified to download the update. The user can also check to automatically install the update at a later time (or set to be notified again later).	

FDP_ACC.2 /Permissions **FDP_ACF.1/ Permissions**	Devices shall provide access controls (permissions) to components on the system and provide the user with the opportunity to grant or block access to these components to any application or if permissions are specifically granted by the device (i.e. in the operating system the application has specific privileges already). This is divided into what the user can explicitly control and what the OS controls.	Supported? (Y/N)
	Document the following: ● List of user permissions (i.e. camera, microphone, contacts) ● List of OS permissions (i.e. Device ID, system permissions, sockets/IPC, files) ● How permissions are granted (including for pre-installed applications where it is granted without user interaction) ● How access is determined (allow/block) ● SELinux is the basis for these permissions and controls	
	FDP_ACC.2.1/Permissions The TSF shall enforce the [*PERMISSIONS POLICY*] on [● *SUBJECTS: APPS, PROCESSES;* ● *USER OBJECTS: [camera, microphone, location, contacts, calendar, call log, stored pictures, text messages, the list of installed apps];*	***Y***

	• *MANUFACTURER OBJECTS: DEVICE ID, SYSTEM PERMISSIONS, FILES (INCLUDING INDIVIDUAL APP DATA), [secure sockets, IPC];* • *OPERATIONS: READ, WRITE, EXECUTE*]. **FDP_ACC.2.2/Permissions** The TSF shall ensure that all operations between any subject controlled by the TSF and any object controlled by the TSF are covered by an access control SFP. **FDP_ACF.1.1/Permissions** The TSF shall enforce the [*PERMISSIONS POLICY*] to objects based on the following: [• *[APPS AND PROCESSES] AND THE OPERATIONS ASSOCIATED WITH THE SUBJECT;* • *THE ACCESS CONTROL LIST ASSOCIATED WITH THE OBJECT BEING REQUESTED*]. **FDP_ACF.1.2/Permissions** The TSF shall enforce the following rules to determine if an operation among controlled subjects and controlled objects is allowed: [• *THE SUBJECT, OR A GROUPING THE SUBJECT IS MAPPED TO, IS EXPLICITLY GRANTED PERMISSION BY THE USER OR TOE MANUFACTURER TO THE USER OBJECT IN THE ACCESS CONTROL LIST***].** **FDP_ACF.1.3/Permissions** The TSF shall explicitly authorize access of subjects to objects based on the following additional rules: [• *THE SUBJECT IS GRANTED PERMISSION BY THE TOE MANUFACTURER TO THE MANUFACTURER OBJECT IN THE ACCESS CONTROL LIST*]. **FDP_ACF.1.4/Permissions** The TSF shall explicitly deny access of subjects to objects based on the following additional rules: [• *THE SUBJECT IS EXPLICITLY BLOCKED BY THE USER FROM ACCESSING THE USER OBJECT;* • *THERE IS NO RULE GRANTING THE SUBJECT ACCESS TO THE USER OR MANUFACTURER OBJECT IN THE ACCESS CONTROL LIST*].	
	The TOE provides the following categories of system services to applications. 1. Normal - A lower-risk permission that gives an application access to isolated application-level features,	

	with minimal risk to other applications, the system, or the user. The system automatically grants this type of permission to a requesting application at installation, without asking for the user's explicit approval (though the user always has the option to review these permissions before installing). 2. Dangerous - A higher-risk permission that would give a requesting application access to private user data or control over the device that can negatively impact the user. Because this type of permission introduces potential risk, the system cannot automatically grant it to the requesting application. For example, any dangerous permissions requested by an application will be displayed to the user and require confirmation before proceeding or some other approach can be taken to avoid the user automatically allowing the use of such facilities. 3. Signature - A permission that the system is to grant only if the requesting application is signed with the same certificate as the application that declared the permission. If the certificates match, the system automatically grants the permission without notifying the user or asking for the user's explicit approval. 4. SignatureOrSystem - A permission that the system is to grant only to packages in the Android system image or that are signed with the same certificates. Please avoid using this option, as the signature protection level should be sufficient for most needs and works regardless of exactly where applications are installed. This permission is used for certain special situations where multiple vendors have applications built in to a system image which need to share specific features explicitly because they are being built together. An example of a normal permission is the ability to vibrate the device: android.permission.VIBRATE. This permission allows an application to make the device vibrate, and an application that does not request (or declare) this permission would have its vibration requests ignored. An example of a dangerous privilege would be access to location services to determine the location of the mobile device: android.permission.ACCESS_FINE_LOCATION. The TOE controls access to Dangerous permissions during the running of the application. The TOE prompts the user to review the application's requested permissions (by displaying a description of each permission group, into which individual permissions	

	map, that an application requested access to). If the user approves, then the application is allowed to continue running. If the user disapproves, the devices continues to run, but cannot use the services protected by the denied permissions. Thereafter, the mobile device grants that application during execution access to the set of permissions declared in its Manifest file. An example of a signature permission is the android.permission.BIND_VPN_SERVICE that an application must declare in order to utilize the VpnService APIs of the device. Because the permission is a Signature permission, the mobile device only grants this permission to an application (2nd installed app) that requests this permission and that has been signed with the same developer key used to sign the application (1st installed app) declaring the permission (in the case of the example, the Android Framework itself). An example of a signatureOrSystem permission is the android.permission.LOCATION_HARDWARE, which allows an application to use location features in hardware (such as the geofencing API). The device grants this permission to requesting applications that either have been signed with the same developer key used to sign the Android application declaring the permissions or that reside in the “system” directory within Android (which for Android 4.4 and above, are applications residing in the /system/priv-app/ directory on the read-only system partition). Put another way, the device grants systemOrSignature permissions by Signature or by virtue of the requesting application being part of the “system image”. Additionally, Android includes the following flags that layer atop the base categories. 1. privileged - this permission can also be granted to any applications installed as privileged apps on the system image. Please avoid using this option, as the signature protection level should be sufficient for most needs and works regardless of exactly where applications are installed. This permission flag is used for certain special situations where multiple vendors have applications built into a system image which need to share specific features explicitly because they are being built together. 2. system - Old synonym for 'privileged'. 3. development - this permission can also (optionally) be granted to development applications (e.g., to allow additional location reporting during beta testing).	

4. appop - this permission is closely associated with an app op for controlling access.
5. pre23 - this permission can be automatically granted to apps that target API levels below API level 23 (Marshmallow/6.0).
6. installer - this permission can be automatically granted to system apps that install packages.
7. verifier - this permission can be automatically granted to system apps that verify packages.
8. preinstalled - this permission can be automatically granted to any application pre-installed on the system image (not just privileged apps) (the TOE does not prompt the user to approve the permission).

For older applications (those targeting Android's pre-23 API level, i.e., API level 22 [lollipop] and below), the TOE will prompt a user at the time of application installation whether they agree to grant the application access to the requested services. Thereafter (each time the application is run), the TOE will grant the application access to the services specified during install.

For newer applications (those targeting API level 23 or later), the TOE grants individual permissions at application run-time by prompting the user for confirmation of each permissions category requested by the application (and only granting the permission if the user chooses to grant it).

The Android 16 (Level 36) API (details found here https://developer.android.com/reference/packages) provides services to mobile applications.

These permissions can be tested using an application built using code that can be found at https://github.com/android/security-certification-resources/tree/master/niap-cc/Permissions.

While Android provides a large number of individual permissions, they are generally grouped into categories or features that provide similar functionality.

Below what the user can directly manage through the UI as permissions, Android uses SELinux in enforcing mode for Mandatory Access Control (and all permission policies are implemented as SELinux policies, just exposed in a more user-friendly way). The SELinux policies can be found on the device using the command:

```
adb pull /sys/fs/selinux/policy
```

	This is the compiled policy file that is enforced on the device and contains all the settings. AOSP defines a number of highly dangerous SELinux controls (DAC_OVERRIDE, NET_ADMIN and SYS_ADMIN) and associates them to services that require those controls. These can be found in the AOSP source in the private and public folders.	

FDP_ACC.1 /UserDataA sset **FDP_ACF.1/ UserDataAs set**	Data stored on the device is protected with different classifications (DE/CE). Describe how user data is protected and how it is classified on the system. DE/CE is sufficient to meet the Low/Medium differences. Explain how High can be implemented.	Supported? (Y/N)
	FDP_ACC.1.1/UserDataAsset The TSF shall enforce the [*USER DATA ASSET DECRYPTION POLICY*] on [• *SUBJECTS: THE TSF;* • *OBJECTS: INTERNAL STORAGE (ALL SAVED USER DATA ASSETS), [no other storage];* • *OPERATIONS: DECRYPT*]. **FDP_ACF.1.1/UserDataAsset** The TSF shall enforce the [*USER DATA ASSET DECRYPTION POLICY*] to objects based on the following: [*SUBJECTS: THE TSF, OBJECTS: USER DATA ASSETS, ATTRIBUTES: SENSITIVE LEVEL OF OBJECTS, LOW, MEDIUM, HIGH*]. **FDP_ACF.1.2/UserDataAsset** The TSF shall enforce the following rules to determine if an operation among controlled subjects and controlled objects is allowed: [• *THE TSF IS ALLOWED TO DECRYPT LOW USER DATA ASSETS IF AND ONLY IF THE TOE IS SUCCESSFULLY POWERED ON; AND* • *THE TSF IS ALLOWED TO DECRYPT MEDIUM USER DATA ASSETS IF AND ONLY IF THE TOE IS SUCCESSFULLY POWERED ON AND THE USER IS SUCCESSFULLY AUTHENTICATED DURING THE FIRST AUTHENTICATION AFTER POWER ON; AND* • *THE TSF IS ALLOWED TO DECRYPT HIGH USER DATA ASSETS IF AND ONLY IF THE TOE IS SUCCESSFULLY POWERED ON, THE USER IS*	***Y***

	SUCCESSFULLY AUTHENTICATED AND THE SCREEN OF THE TOE IS NOT LOCKED]. **FDP_ACF.1.3/UserDataAsset** The TSF shall explicitly authorize access of subjects to objects based on the following additional rules: [*NO ADDITIONAL RULES*]. **FDP_ACF.1.4/UserDataAsset** The TSF shall explicitly deny access of subjects to objects based on the following additional rules: [*NO ADDITIONAL RULES*].	
	The device provides different classes of storage for all user data. By default all user data is protected with a key that is derived from the user's password (this is considered Medium protection). The device only supports internal storage. For Low data, an application must be explicitly developed to take advantage of this feature which will allow data saved by the app as DE data to be available on reboot before the user has authenticated. For High data, an application must be explicitly developed to be aware of the lock status of the device to be able to close and lock its data. Android APIs provide the ability to hook into the notification of the lock status to enable this functionality. In all cases, keys that are stored on the device are entangled with the hardware root key (in the case of Low, this is combined with a hardcoded device key to take the place of the user's password). This ensures that data cannot be ported to another device and decrypted as only the device where the data is encrypted will have access to the hardware key used to generate the encryption key.	

Identification and Authentication

FIA_UAU.1 **FIA_UID.1**	When authentication is enabled, some actions may be performed before a successful login. Document what actions are allowed before a successful login. Access to stored data shall not be allowed (i.e. access to CE data)	Supported? (Y/N)

	FIA_UAU.1.1 The TSF shall allow [
	
	• Take screen shots (stored internally)
	• Make emergency calls
	• Receive calls
	• Take pictures (stored internally) - unless the camera was disabled
	• Turn the TOE off
	• Restart the TOE
	• Place TOE into lockdown mode
	• Enable Airplane mode
	• Change the state of Wi-Fi, Bluetooth, Mobile Data (cellular data)
	• Change Battery Saver mode
	• Adjust screen brightness
	• See notifications (note that some notifications identify actions, for example to view a screenshot; however, selecting those notifications highlights the password prompt and require the password to access that data)
	• Configure sound, vibrate, or mute
	• Set the volume (up and down) for ringtone
	• Change keyboard input method
	• Change live captions
	• Access notification widgets (without authentication):
	  ○ Flashlight toggle
	  ○ Do not disturb toggle
	  ○ Auto rotate toggle
	  ○ Sound (on, mute, vibrate)
	  ○ Night light filter toggle
	
	] on behalf of the user to be performed before the user is authenticated.
	
	FIA_UAU.1.2 The TSF shall require the user to be successfully authenticated before allowing any other TSF-mediated actions on behalf of that user.
	
	FIA_UID.1.1 The TSF shall allow [
	
	• Take screen shots (stored internally)
	• Make emergency calls
		Y

	• Receive calls
	• Take pictures (stored internally) - unless the camera was disabled
	• Turn the TOE off
	• Restart the TOE
	• Place TOE into lockdown mode
	• Enable Airplane mode
	• Change the state of Wi-Fi, Bluetooth, Mobile Data (cellular data)
	• Change Battery Saver mode
	• Adjust screen brightness
	• See notifications (note that some notifications identify actions, for example to view a screenshot; however, selecting those notifications highlights the password prompt and require the password to access that data)
	• Configure sound, vibrate, or mute
	• Set the volume (up and down) for ringtone
	• Change keyboard input method
	• Change live captions
	• Access notification widgets (without authentication):
	  ○ Flashlight toggle
	  ○ Do not disturb toggle
	  ○ Auto rotate toggle
	  ○ Sound (on, mute, vibrate)
	  ○ Night light filter toggle
	
	] on behalf of the user to be performed before the user is identified.
	
	FIA_UID.1.2 The TSF shall require each user to be successfully identified before allowing any other TSF-mediated actions on behalf of that user.	

	The following actions can be taken before the user has authenticated to the device: • Take screen shots (stored internally) • Make emergency calls • Receive calls • Take pictures (stored internally) - unless the camera was disabled • Turn the TOE off • Restart the TOE • Place TOE into lockdown mode • Enable Airplane mode • Change the state of Wi-Fi, Bluetooth, Mobile Data (cellular data) • Change Battery Saver mode • Adjust screen brightness • See notifications (note that some notifications identify actions, for example to view a screenshot; however,	

	selecting those notifications highlights the password prompt and require the password to access that data)
	• Configure sound, vibrate, or mute
	• Set the volume (up and down) for ringtone
	• Change keyboard input method
	• Change live captions
	• Access notification widgets (without authentication):
	  ○ Flashlight toggle
	  ○ Do not disturb toggle
	  ○ Auto rotate toggle
	  ○ Sound (on, mute, vibrate)
	  ○ Night light filter toggle
	
	All other actions require the user to be authenticated.	

FIA_UAU.5/ Local	Multiple methods of authentication may be supported. Describe the methods of authentication that are provided including any biometric modalities or external devices that may be supported. Describe the ordering for how multiple methods may be used at one time (for example if fingerprint is enabled, how does the device decide whether to ask for the fingerprint or PIN).	Supported? (Y/N)
	FIA_UAU.5.1/Local The TSF shall provide [*PASSWORD, PIN* and pattern, BAF in accordance with the PP-Module for Biometrics Authentication] to support user authentication. **FIA_UAU.5.2/Local** The TSF shall authenticate any user's claimed identity according to the [following rules: To authenticate unlocking the device immediately after boot (first unlock after reboot): • User passwords are required after reboot to unlock the user's Credential encrypted (CE files) and keystore keys. Biometric authentication is disabled immediately after boot. To authenticate unlocking the device after device lock (not following a reboot): • The TOE verifies user credentials (password or fingerprint) via the gatekeeper or fingerprint trusted application (running inside the Trusted Execution Environment, TEE), which compares the entered credential to a derived value or template.	***Y***

	To change protected settings or issue certain commands: ● The TOE requires password after a reboot, when changing settings (Screen lock, Fingerprint, and Smart Lock settings), and when factory resetting.].	
	The device supports the use of a password, PIN, pattern and fingerprint for authentication. When the device is first started (or restarted), the biometric cannot be used for authentication until after a password, pattern or PIN has been entered first. The fingerprint is offered as the default authentication mechanism as it can be entered without fully waking the screen.	

FIA_UAU.5/Peer

This is for functionality that is provided with the device from the developer and does not include apps that may be downloaded by the user after purchase.

FIA_UAU.5/ Peer	When connecting to a peer device or a service, the user shall successfully authenticate before the connection is allowed.	Supported? (Y/N)
	Document the methods for connecting to: ● peer-to-peer device connections ○ Bluetooth pairing methods are handled in FTP_ITC_EXT.1/BT and not necessary here ● Other services ○ For example backup/sync services using a trusted server ○ Screen mirroring/sharing For other services, specify what is allowed after the successful pairing	
	FIA_UAU.5.1/Peer The TSF shall provide **support to use** [[QR codes, NFC labels, using a common user account to a remote service on both devices] **for** ~~to support~~ ~~user~~ authentication **to peer devices before allowing any actions on behalf of the user**. **FIA_UAU.5.2/Peer** The TSF shall authenticate any ~~user's~~ **peer device's** claimed identity according to the [QR codes can provide Wi-Fi information, NFC labels can provide pairing information for Bluetooth connections, Accounts on the device can be used to sign in to remote services].	***Y***

	The device provides the following ways to authenticate connections to peer devices/services: ● QR codes - these can be used to add Wi-Fi information to connect to an AP ○ Note that QR codes can be read automatically by the camera, but by default are handed to the web browser and are not tied to the device itself ● NFC labels - these can be used to provide connectivity for pairing to Bluetooth devices ● Apps can register accounts (seen in Settings -> Passwords & accounts) that can be used to connect to remote services (such as a mail server or other cloud service). These require the user to enter a valid username/password combination on the device that matches an account on the service being connected to.	

FIA_UAU.6	The user needs to be re-authenticated to perform specific actions (this is specifically after the initial authentication to unlock the device after a bootup/startup).	
	Document what actions require authentication: ● Unlocking the device after it has become locked (mandatory) ● Change of authentication data (any change) (mandatory) ● Other conditions?	Supported? (Y/N)
	FIA_UAU.6.1 The TSF shall re-authenticate the user under the conditions [● *ATTEMPTED CHANGE OF ANY USER AUTHENTICATION FACTOR;* ● *ATTEMPTED UNLOCKING OF A LOCKED TOE;* ● *[*no other conditions*]*].	
	The device requires the user to enter their password or supply their biometric in order to unlock the device. Additionally the device requires the user to confirm their current password when accessing the Settings -> Security -> Screen lock menu in the user interface. Only after entering their current user password can the user then elect to change their password.	***Y***

FIA_UAU.7	The user should see only limited feedback during authentication is in progress	Supported? (Y/N)
	Describe what feedback is provided to the user during authentication, such as password/PIN display (how long before masking).	
	FIA_UAU.7.1 The TSF shall provide only [*BRIEF FEEDBACK ABOUT THE ENTERED CREDENTIALS*] to the user while the authentication is in progress.	
	The device allows the user to enter the user's password from the lock screen. The device will display the most recently entered character of the password briefly or until the user enters the next character in the password, at which point the device obscures the character by replacing the character with a dot symbol. Further, the device provides no feedback other than whether the fingerprint unlock attempt succeeded or failed.	***Y***

FIA_SOS.1	The user shall be able to set credentials of a minimum strength	Supported? (Y/N)
	Document the minimum/maximum requirements for password/PIN/pattern settings for the user and specify the character set used for a password.	
	FIA_SOS.1.1 The TSF shall provide a mechanism to verify that secrets meet [• *FOR PASSWORD AND PIN: LENGTH 4 OR MORE FROM A DEFINED CHARACTER SET;* • *FOR PATTERNS: CONSISTS OF AT LEAST 4 FROM A SET OF AT LEAST 9 AVAILABLE POINTS, WHERE EACH POINT SHALL ONLY BE USED ONCE*].	
	The minimum length for any password or PIN is 4 characters/numbers. Passwords consist of basic Latin characters (upper and lower case, numbers, and the following special characters: ! @ # $ % ^ & * ( ) [= + - _ ` ~ \ |] } [{ ‘ “ ; : / ? . > , < The maximum length for a password or PIN is 16 characters/numbers. For a pattern the user must trace a pattern that touches at least 4 individual points from the 9 available.	***Y***

<table>
<tr><td rowspan="4">FIA_AFL.1</td><td>When repeated authentication attempts are detected, the device should deter continued attempts. This includes all available authentication methods.</td><td rowspan="2">Supported? (Y/N)</td></tr>
<tr><td>Document what actions are taken when some number of attempts have been detected (between 3-10).</td></tr>
<tr><td>FIA_AFL.1.1 The TSF shall detect when [<u>AN INTEGER BETWEEN 3 AND 10</u>] unsuccessful authentication attempts occur related to [Password, PIN, Pattern, BAF in accordance with the PP-Module for Biometrics Authentication].

FIA_AFL.1.2 When the defined number of unsuccessful authentication attempts has been [<u>MET</u>], the TSF shall [

● Block the biometric use after 5 unsuccessful attempts
● Require a 30 second delay after 5 unsuccessful non-biometric attempts, and then again after 10 total attempts
● Additional attempts past 11 will have at least a 30 second delay on every attempt (eventually growing to a day between attempts)].</td><td rowspan="2">Y</td></tr>
<tr><td>The device maintains in persistent storage the number of failed authentication attempts since the last login.

The Gatekeeper code for calculating the delay is here (written below based on the number of attempts taken).

<pre>[0, 4] -> 0
5 -> 30
[6, 10] -> 0
[11, 29] -> 30
[30, 139] -> 30 * (2^((x - 30)/10))
[140, inf) -> 1 day</pre>
Every 5 consecutive failed attempts the device will force the user to wait for a period of time before further attempts (starting at 30 seconds and then doubling in time). Every 10 attempts it also requires the device to reboot to perform further attempts.

Biometric authentication can be attempted up to 5 times before the biometric is disabled and the user can only use a password, PIN or pattern to authenticate.

When biometrics are enabled, they can be used once the screen wakes up. Entering a password, PIN or pattern requires an additional swipe of the screen to activate the sign in dialog to enter the credential.</td></tr>
</table>

	Upon successful authentication the failure count is reset to 0.	

Security Management

FMT_MSA.1 /Permissions **FMT_MSA.3 /Permissions**	The user shall be able to manage the user permissions on the available objects. The default policy if no permission is assigned should be restrictive. Document what the user can do in setting permissions for access (approve/reject persistently or temporarily, etc)	Supported? (Y/N)
	FMT_MSA.1.1/Permissions The TSF shall enforce the [*PERMISSIONS POLICY*] to restrict the ability to [approve persistently, approve temporarily, reject persistently, reject temporarily, modify] the security attributes [*LIST OF USER OBJECTS] to [THE CURRENT USER FOR THEIR OWN PERMISSIONS]*. **FMT_MSA.3.1/Permissions** The TSF shall enforce the [*PERMISSIONS POLICY*] to provide [RESTRICTIVE] default values for security attributes that are used to enforce the SFP. **FMT_MSA.3.2/Permissions** The TSF shall allow the [*CURRENT USER FOR THEIR OWN PERMISSIONS*] to specify alternative initial values to override the default values when an object or information is created.	
	The user can manage the permissions for an application based on 9 different groups of permissions. Applications that are preloaded (or that are part of the system) are assigned permissions by Google to ensure the functionality of the device (but are not assigned more permissions than are necessary for the function of the app). Apps that are installed by the user are assigned no permissions during the installation. The permissions that are needed by the app (some apps may not require any permissions) will be requested on first use. At that time the user can choose to allow continued access, allow access only for that time, or to reject granting access to the permissions. If the user granted access only once, the user will continue to be prompted every time the app is run to allow access to that permission. Permissions for an app can be modified after they have initially been set through Settings -> Apps -> <app in question>	***Y***

	Here the user can choose to change the permissions that have been granted (or rejected).	

FMT_SMF.1 /Authentication	The user shall be able to specify and later change their authentication credentials	Supported? (Y/N)
	Document how the user can set and change the password/PIN/pattern/biometric	
	FMT_SMF.1.1/Authentication The TSF shall be capable of performing the following management functions: [• entering an initial or changing (including removal of) the KAF].	***Y***
	The user can change their authentication credentials through Settings -> Security & privacy -> Device lock From here the user can choose Screen lock or Fingerprint Unlock to change their credentials (set, remove, modify, etc). Setting a fingerprint will require the user to also set a password, PIN or pattern.	

FMT_SMF.1 /Permissions	The user shall be able to manage the permissions provided to apps and some system services	Supported? (Y/N)
	• Document how the user can view, grant and revoke permissions for any app	
	FMT_SMF.1.1/Permissions The TSF shall be capable of performing the following management functions: [• *VIEW PERMISSIONS GRANTED TO AN APP; AND* • *GRANT/REVOKE PERMISSION TO/FROM AN APP OR PROCESS TO HAVE READ AND/OR WRITE ACCESS TO A USER OBJECT*].	***Y***
	See FMT_MSA.1/3 answers	

	The user shall be able to manage various settings on the device	

FMT_SMF.1 /UserControls	• Document the settings for accessibility service, notifications • Document how the user can set the default USB mode when connecting to a computer • Removable media encryption	Supported? (Y/N)
	FMT_SMF.1.1/UserControls The TSF shall be capable of performing the following management functions: [• *GRANT/REVOKE PERMISSION TO/FROM AN APP OR PROCESS TO HAVE ACCESS TO ACCESSIBILITY SERVICE; AND* • *GRANT/REVOKE PERMISSION TO/FROM AN APP OR PROCESS TO HAVE ACCESS TO DEVICE NOTIFICATION; AND* • *[*charge only mode by default, file transfer mode*] WHEN THE TOE IS CONNECTED VIA THE WIRED CHARGING INTERFACE TO ANOTHER DEVICE; AND* • *[*no support for removable media*]; and* • *[*no other functions*]*].	
	The user can manage apps that can access the accessibility service through: Settings -> Accessibility The user can manage the notifications (which apps can provide notifications, access to read notifications, whether they can be seen on the lock screen) through: Settings -> Notifications The user can manage USB connectivity to another device (such as a PC) through the USB preferences. By default the selection is to not transfer data over the connection. This option becomes available when the device is connected to another device, at which point these settings can be adjusted for the connection. The device does not support removable media	***Y***

FMT_SMF.1 /Privacy	The user shall be able to change the alias used as an identifier for ads and developers	Supported? (Y/N)
	Document how the user can change the alias (or disable it).	
	FMT_SMF.1.1/Privacy The TSF shall be capable of performing the following management functions: [	***Y***

	• change or reset the privacy aliases; • block the creation/use of a unique ID for advertising (no personalized tracking)].	
	The user can access the advertising controls in: Settings -> Security & privacy -> More privacy settings -> Ads Here the user can reset the advertising ID or delete it completely. If the ID is deleted then personalized tracking can be disabled. The user generally does not have access to identifiers that have been requested by an app. Some apps may provide this information internally, but this is a developer choice. If the user wants to reset the identifier (or delete it), they can delete the app (or clear all storage for the app, which would remove all associated app data). This will delete the local identifier such that the user would be able to have a new identifier created the next time the app is run.	

FMT_SMF.1/APP_Update	The user shall be able to manage applications on the device (update/uninstall)	Supported? (Y/N)
	Describe how application updates can be initiated and how apps (including pre-installed) can be uninstalled.	
	FMT_SMF.1.1/APP_Update The TSF shall be capable of performing the following management functions: [~~assignment:~~ • *SPECIFY TO [notify the user without downloading, automatically install] WHEN AN AUTOMATIC CHECK TO THE ADP HAS FOUND AN UPDATE; AND* • *INITIATE AN IMMEDIATE CHECK FOR UPDATE; AND* • *INITIATE AN UPDATE OF THE APP (IF AVAILABLE); AND* • *DISPLAY THE VERSION NUMBER OF THE APP; AND* • *UNINSTALL DOWNLOADED APPS (INCLUDING APPS DOWNLOADED AS PART OF THE SETUP PROCESS)*].	***Y***
	Through the Play Store the user can manage the apps they download to the device. The Play Store allows the user to install apps, check for updates, and uninstall apps. The Play Store checks for updates to the installed apps roughly once per day to see if there are new versions to be installed. In general the apps	

	will be updated automatically, though if there are permission changes to the new version, the user will be required to approve the installation of the new version. The user can check the version number of any app on the device through: Settings -> Apps -> <app in question> The app version will be displayed at the bottom of the information. The user can also uninstall any app (that can be uninstalled) in the same Settings page by clicking the Uninstall button for the app.	

FMT_SMF.1 /SSW_Update	The user shall be able to check for system software updates Describe how the user can check for new system software updates and how they can be installed.	Supported? (Y/N)
	FMT_SMF.1.1/SSW_Update The TSF shall be capable of performing the following management functions: [~~assignment:~~ • • *SPECIFY TO [notify the user without downloading, automatically install] WHEN AN AUTOMATIC CHECK TO THE TRUSTWORTHY UPDATE SOURCE HAS FOUND AN UPDATE; AND* • *INITIATE AN IMMEDIATE CHECK FOR UPDATE; AND* • *INITIATE AN UPDATE OF THE SYSTEM SOFTWARE (IF AVAILABLE); AND* • *PROVIDE THE STATUS OF THE UPDATE PROCESS AND THE RESULTS OF THE UPDATE; AND* • *[delay temporarily] AUTOMATIC INSTALLATION OF SYSTEM SOFTWARE UPDATES; AND* • *DISPLAY THE VERSION NUMBER OF THE SYSTEM SOFTWARE*].	***Y***
	The user can check for new updates by going to: Settings -> System -> System Update This location also shows the installation status/progress of the update (until the reboot to switch to the updated version).	

	The user is able to temporarily delay the installation of an update when presented with the option to to download it. After 3 delays the user will be forced to update the device. This screen also shows the current version of the system software, stating the Android version and the security update version.	

Privacy

FPR_PSE.1	Advertisers and app developers shall be provided a unique device identifier that can be modified by the user.	Supported? (Y/N)
	Describe how the unique identifier is provided and how this can be changed by the user.	
	FPR_PSE.1.1/Advertisers The TSF shall ensure that [*AD NETWORKS*] are unable to ~~determine~~ **access** the ~~real user name~~ **Device ID** bound to [*THE TOE (HARDWARE PLATFORM)*] **according to the Permissions Policy**. **FPR_PSE.1.2/Advertisers** The TSF shall be able to provide [*AT LEAST ONE UNIQUE*] alias**(**es**)** of the ~~real user name~~ **Device ID** to [*AD NETWORKS*]. **FPR_PSE.1.3/Advertisers** The TSF shall [DETERMINE AN ALIAS FOR A DEVICE ID]~~ and verify that it conforms to the [assignment: *alias metric*]~~. **FPR_PSE.1.1/APP_Dev** The TSF shall ensure that [*APP DEVELOPERS*] are unable to ~~determine~~ **access** the ~~real user name~~ **Device ID** bound to [*THE TOE (HARDWARE PLATFORM)*] **according to the Permissions Policy**. **FPR_PSE.1.2/APP_Dev** The TSF shall be able to provide [*AT LEAST ONE UNIQUE*] alias**(**es**)** of the ~~real user name~~ **Device ID** to [*EACH APP DEVELOPER*]. **FPR_PSE.1.3/APP_Dev** The TSF shall [PROVIDE AN ALIAS FOR A DEVICE ID]~~ and verify that it conforms to the [assignment: *alias metric*]~~ **upon request by the App developer**.	***Y***
	The device provides a single identifier for advertisers to use. This identifier is not tied to the hardware (it is not derived from any hardware identifiers) and is randomly generated. Advertisers are not able to access unique hardware identifiers through the	

	access control permissions on access to them (such as the IMEI). The advertiser identifier string can be seen by the user in: Settings -> Security & privacy -> More privacy settings -> Ads App developers can have unique identifiers generated for them by the system to enable identification of the device tied to the user within their services. The unique identifier for the app is randomly generated and not tied to the hardware unique identifiers in any way. Each app developer can request an identifier (a developer with multiple apps may utilize the same identifier across all apps, that is a developer choice). An app developer can use the API randomUUID to request a random identifier.	

Protection of the TSF

FPT_PHP.3	OEMs and ODMs shall provide a hardware-backed key store implemented within a SEE, Secure Element or equivalent solution. Plaintext private key material shall not exist outside of the hardware-backed key store.	Supported? (Y/N)
	Provide documentation on the hardware-backed credential storage capabilities of the device, as well as the specific configurations.	
	FPT_PHP.3.1 The TSF shall resist [*to:* ● *READ OR MODIFY THE DUK; AND* ● *READ OR MODIFY [*no keys (keys are not stored unencrypted)*]; AND* ● *MODIFY [*hashes of keys*] USED TO VERIFY THE INTEGRITY OF THE TSF IN FPT_TST.1; AND* ● *MODIFY [*hashes of keys*] USED TO VERIFY THE INTEGRITY AND AUTHENTICITY OF UPDATES TO THE TSF IN FCS_COP.1/ASYMMETRIC; AND* ● *READ OR MODIFY [*no other keys*]*]; to the **[***HARDWARE BASED SECURE ENVIRONMENT OF THE TSF***]** by responding automatically such that ~~the SFRs are always enforced~~ **it is impossible to read or modify this data and/or key(s)**.	***Y***
	See the Tables in the KMD for information about keys and where they are stored The device provides multiple hardware-based storage locations for keys to maintain confidentiality and integrity. The DUK (listed as the REK in the KMD) is stored in one-time fuses in the AP itself. These are never read directly by anything outside a security subsystem included in the AP. The DUK is used in the TEE (Trusty 13261797), but not directly. The TEE requests actions related to the key to be performed (such as to encrypt or decrypt another key) to the AP subsystem which then performs the action and returns the encrypted result. This ensures that the DUK cannot be accessed outside the AP hardware. Direct reading of the DUK would require knowing the	

	physical location within the AP and non-destructively being able to read the individual fuses. Android provides the Keystore to securely store all keys inside the TEE. All keys will be encrypted with a key tied to the DUK, and where appropriate, the user's password (some keys are only protected by the DUK, such as Wi-Fi keys, so they are accessible before the user has authenticated, and is determined by the app/process storing the key). Not all keys are stored and may be derived as needed (such as those tied to the user's credentials). The device also provides support for the StrongBox Keystore. StrongBox utilizes the Titan M2 chip to provide a separate hardware keystore (similar to a secure element, but more like a secure processor unit) which has been certified. Apps and processes can use this to store keys instead of just placing them into the TEE Keystore. Access to StrongBox is through the normal Keystore API with a separate flag to note that the key must be stored in dedicated hardware. All boot keys have SHA-256 hashes fused into the AP to verify the signatures on the boot images. The kernel signing key is embedded into the Android bootloader (which is signed with a boot key). The dm-verity key that protects the /system and /vendor partitions is stored in the kernel image (protected by the kernel signing key which is protected by the bootloader signing key hash). The OTA key is similarly stored in the kernel image (for recovery mode) and in the /system partition (for normal OTA updates).	

FPT_FLS.1	A failure in the system software update shall not expose user data.	Supported? (Y/N)
	Describe how a failure of the update process is managed. For example, an automatic rollback, or some other process which would allow the user to restore the system but which would not expose any encrypted user data.	
	FPT_FLS.1.1 The TSF shall preserve a secure state when the following types of failures occur: [*FAILURE OF THE UPDATE_THE_TOE_SOFTWARE OPERATION IN FDP_ACF.1/SSW_UPDATE*].	***Y***
	All user data on the device is encrypted. Except for data that is not encrypted with the user's credentials (DE data), user data cannot be decrypted without a successful user authentication.	

	So a failure to update the system software cannot unlock the user data as the new system would not be able to properly start (the failure). The device maintains two boot slots for the system. Only one is active at a time, and the alternate slot is used to update the device. If the update process fails at any point (such as during the write operation or upon reboot the system cannot successfully start and the user authenticates), the system will revert to the original boot slot and return to the previous version of Android (the version running at the time the update was downloaded).	

FPT_TST.1	During the device start-up process, the integrity of the system software shall be verified.	Supported? (Y/N)
	Different tests are allowed for different types of checks. For example: ● Cryptographic algorithms can run self-tests (RBG can verify output) ● Bootloader and other system checks using hash trees, signatures (or hashes)	
	FPT_TST.1.1 The TSF shall run a suite of self-tests **and integrity verification** [DURING INITIAL START-UP] to demonstrate the correct operation of [*[*cryptographic checks on the algorithm tests in BoringSSL, integrity test of BoringSSL and entropy health checks from SP800-90B on the hardware noise source*]*] BY SELF TESTS AND THE BOOTLOADER, MAIN OS KERNEL [SEE, executable code stored in /system and /vendor partitions] BY INTEGRITY, WHERE INTEGRITY IS VERIFIED BY [a hash of an asymmetric key]. **FPT_TST.1.2** The TSF shall provide authorised users with the capability to verify the integrity [*NO DATA*]. **FPT_TST.1.3** The TSF shall provide authorised users with the capability to verify the integrity of [*NONE*].	***Y***
	The device runs a number of self-tests on specific components to ensure they are properly functioning during the start-up. Failure of these tests will cause the boot sequence to halt and a Boot Failure message will be shown. The tests run are: ● Algorithm tests on BoringSSL (AES, SHA, HMAC, DRBG, ECDSA, RSA ECDH) ● Self-test on BoringSSL (module integrity after load)	

	• Entropy health tests (from the SoC) based on SP 800-90B These tests are run every time the component is started. For example, each time BoringSSL is loaded into memory it will perform the algorithm and self tests. The entropy health tests are started when the device powers on and are then run continuously thereafter. The device verifies the integrity of the following components during the start-up: • Bootloader • OS kernel • TEE (Trusty) • Executable code stored in /system and /vendor partitions All these are verified by checking the signature using the hash of an asymmetric key that is fused into the SoC during manufacturing. The code stored in the /system and /vendor partitions is checked using dm-verity, where the tree is verified to the hash. Dm-verity can correct some errors automatically (depends on the size of the error). Other than errors that can be corrected automatically, any failure of a match will cause the device to halt the boot process.	

FPT_RCV.2	The device shall support a maintenance mode to enable recovery from malicious/persistent software.	Supported? (Y/N)
	Describe the process by which malicious software may be removed automatically (where possible), and if this isn't possible, how the user may do so (for example, safe boot, manually re-installing the system software or a factory reset)	
	FPT_RCV.2.1 When automated recovery from [*DETECTION OF A MALEVOLENT PERSISTENT PRESENCE BY FPT_TST.1 OR AN UPDATE FAILURE BY FDP_ACF.1/SSW_UPDATE*] is not possible, the TSF shall enter a maintenance mode where the ability to return to a secure state is provided. **FPT_RCV.2.2** For [*DETECTION OF A MALEVOLENT PERSISTENT PRESENCE BY FPT_TST.1 OR AN UPDATE FAILURE BY FDP_ACF.1/SSW_UPDATE*], the TSF shall ensure the return of the TOE to a secure state using automated procedures.	***Y***

	For detection of issues in the /system and /vendor partitions (such as an attempt to write malicious code or make other malicious changes to code stored there), most errors will be automatically corrected by the dm-verity check during the start-up (the error will be detected and corrected automatically). For larger failures, such as changes to other code on the system (i.e. not in the /system or /vendor partitions), the device will boot into a recovery state where-in the user can manually reload known-good firmware onto the device (factory images can be found at: https://developer.android.com/about/versions/15/download).	

Trusted Path/Channels

FTP_ITC_EXT.1/BT	The device shall support at least Bluetooth Core Specification v4.1	Supported? (Y/N)
	Document the Bluetooth support on the device Describe the process for regenerating ECDH key pairs with paired devices	
	FTP_ITC_EXT.1.1/BT The TSF shall use [*BLUETOOTH® CORE SPECIFICATION THAT CONFORMS TO [v5.2 [15]]*] to provide a communication channel between itself and another trusted IT product that is logically distinct from other communication channels and provides assured identification of its end points and protection of the channel data from modification or disclosure. **FTP_ITC_EXT.1.2/BT** The TSF shall permit [the TSF, another trusted IT product] to initiate communication via the trusted channel. **FTP_ITC_EXT.1.3/BT** The TSF shall initiate communication via the trusted channel for [*CONNECTIONS TO BLUETOOTH DEVICES*]. **FTP_ITC_EXT.1.4/BT** The protocol used by the communications channel shall support the following requirements: [• *REQUIRE EXPLICIT USER AUTHORISATION BEFORE PAIRING; AND* • *USE SECURE SIMPLE PAIRING AND SECURE CONNECTIONS FOR PAIRING; AND* • *NOT ALLOW MORE THAN ONE BLUETOOTH CONNECTION TO THE SAME BLUETOOTH DEVICE ADDRESS; AND* • *GENERATE NEW ECDH PUBLIC/PRIVATE KEY PAIRS EVERY [on every connection between the devices]*].	***Y***
	The device has been qualified according to the Bluetooth 5.2 specification. Pairing is not allowed without explicit authorization from the user (this cannot be programmatically entered but must be input directly from the user). If the device is already paired with a peer, a second peer attempting to connect with the same BD_ADDR will be rejected (so a device that has been made to look like the first peer will be blocked from connecting while the original device is already connected).	

	Each time the device is connected to a peer over Bluetooth a new ECDH key pair will be created for the connection.	

FTP_ITC_EXT.1/HTTPS **FTP_ITC_EXT.1/TLS**	The device shall support TLS 1.2 or higher Document the following: • versions of TLS that are supported (1.2 & 1.3 expected) • IETF RFC 5280 support for certificate revocation checking • What happens if a certificate is determined to be invalid • What TLS ciphersuites are supported (for apps/websites to use)	Supported? (Y/N)
	FTP_ITC_EXT.1.1/HTTPS The TSF shall use [*HTTPS THAT CONFORMS TO IETF RFC 2818 [5]*] to provide a communication channel between itself and another trusted IT product that is logically distinct from other communication channels and provides assured identification of its end points and protection of the channel data from modification or disclosure. **FTP_ITC_EXT.1.2/HTTPS** The TSF shall permit [THE TSF] to initiate communication via the trusted channel. **FTP_ITC_EXT.1.3/HTTPS** The TSF shall initiate communication via the trusted channel for [*COMMUNICATION WITH A TRUSTED IT PRODUCT*]. **FTP_ITC_EXT.1.4/HTTPS** The protocol used by the communications channel shall support the following requirements: [*USE TLS AS SPECIFIED IN FTP_ITC_EXT.1/TLS TO IMPLEMENT HTTPS*]. **FTP_ITC_EXT.1.1/TLS** The TSF shall use [*TLS THAT CONFORMS TO [*TLS v1.2 [6], TLS v1.3 [10]*] to provide a communication channel between itself and another trusted IT product that is logically distinct from other communication channels and provides assured identification of its end points and protection of the channel data from modification or disclosure. **FTP_ITC_EXT.1.2/TLS** The TSF shall permit [*the TSF*] to initiate communication via the trusted channel. **FTP_ITC_EXT.1.3/TLS** The TSF shall initiate communication via the trusted channel for [*COMMUNICATION WITH A TRUSTED IT PRODUCT*].	***Y***

	FTP_ITC_EXT.1.4/TLS The protocol used by the communications channel shall support the following requirements: [• *SUPPORT X.509V3 CERTIFICATES FOR MUTUAL AUTHENTICATION; AND* • *DETERMINE VALIDITY OF THE PEER CERTIFICATE BY CERTIFICATE PATH, EXPIRATION DATE AND REVOCATION STATUS ACCORDING TO IETF RFC 5280 [7]; AND* • *NOTIFY THE TSF AND [*not establish the connection*] IF THE PEER CERTIFICATE IS DEEMED INVALID; AND* • *SUPPORTS THE FOLLOWING CIPHER SUITES [* ○ TLS_RSA_WITH_AES_256_GCM_SHA384 (IETF RFC 5288 [8]), ○ TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (IETF RFC 5289 [9]), ○ TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (IETF RFC 5289 [9]), ○ TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (IETF RFC 5289 [9]), ○ TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (IETF RFC 5289 [9]), ○ TLS_AES_128_GCM_SHA256 as defined in RFC 8446 ○ TLS_AES_256_GCM_SHA384 as defined in RFC 8446 ○ TLS_CHACHA20_POLY1305_SHA256 as defined in RFC 8446*]*].	
	The device supports TLS v1.2 and TLS v1.3 (TLS v1.3 is the default). TLS is available directly via Android APIs (such as for an app to communicate to a server) or through HTTPS connections (such as when using a web browser. The device supports mutual authentication using certificates, and can check the validity of the peer certificate according to RFC 5280. Android supports the ability for an app to check the certificate revocation status using OCSP. The app must implement the APIs to enable the check and determine the action to take if the certificate is found to be revoked (Android does not perform any action based on the information itself other than notify the app requesting the check of the state).	

	The device does not act as a server so TLS connections must be initiated by the device (a request from a server to switch to a TLS connection will cause the device to start one) as opposed to accepting incoming connections that are not responses to a request originating from the device. The following ciphersuites are supported: • TLS_RSA_WITH_AES_256_GCM_SHA384 as defined in RFC 5288, • TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 as defined in RFC 5289, • TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 as defined in RFC 5289, • TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 as defined in RFC 5289, • TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 as defined in RFC 5289 • TLS_AES_128_GCM_SHA256 as defined in RFC 8446 • TLS_AES_256_GCM_SHA384 as defined in RFC 8446 • TLS_CHACHA20_POLY1305_SHA256 as defined in RFC 8446	

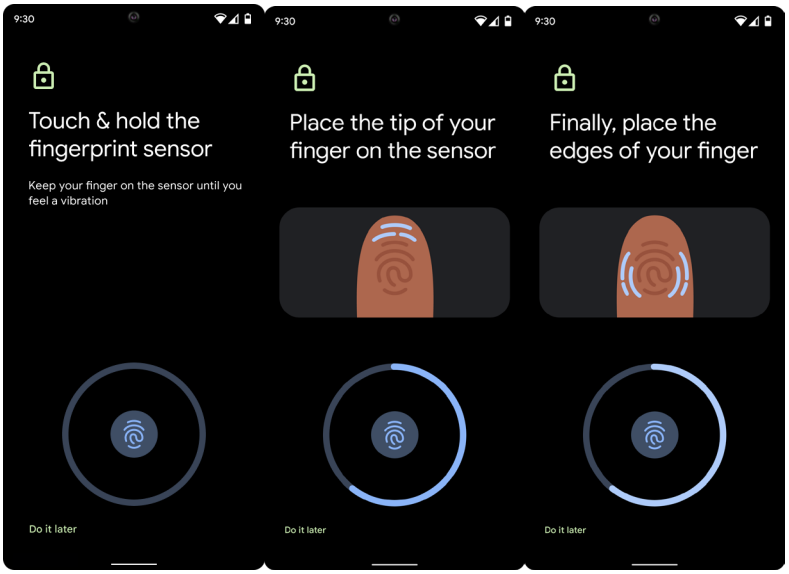
FTP_ITC_EXT.1/WLAN	The device shall support IEEE 802.11-2012, 802.1X & EAP-TLS connectivity	Supported? (Y/N)
	Document the Wi-Fi support on the device including • Key lengths supported • TLS 1.2 or higher support • IETF RFC 5280 support for certificate revocation checking • What happens if a certificate is determined to be invalid • What TLS ciphersuites are supported • MAC address randomization process (frequency)	
	FTP_ITC_EXT.1.1/WLAN The TSF shall use [*WLAN THAT CONFORMS TO 802.11-2012 [16]*] to provide a communication channel between itself and another trusted IT product that is logically distinct from other communication channels and provides assured identification of its end points and protection of the channel data from modification or disclosure. **FTP_ITC_EXT.1.2/WLAN** The TSF shall permit [*the TSF*] to initiate communication via the trusted channel. **FTP_ITC_EXT.1.3/WLAN** The TSF shall initiate communication via the trusted channel for [*COMMUNICATION WITH THE TRUSTED IT PRODUCT THROUGH WLAN CHANNEL*].	***Y***

	FTP_ITC_EXT.1.4/WLAN The protocol used by the communications channel shall support the following requirements: [
	
	• *GENERATE SYMMETRIC KEYS ACCORDING TO [*PRF-384 with key length 128 bit, PRF-704 with key length 256 bit*]; AND*
	• *USES [TLS V1.2 [6]]; AND*
	• *SUPPORTS THE FOLLOWING CIPHER SUITES [selection:*
	  ○ *TLS_RSA_WITH_AES_256_GCM_SHA384 (IETF RFC 5288 [8]),*
	  ○ *TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (IETF RFC 5289 [9]),*
	  ○ *TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (IETF RFC 5289 [9]),*
	  ○ *TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (IETF RFC 5289 [9]),*
	  ○ *TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (IETF RFC 5289 [9])]*
	• *RANDOMLY GENERATE A NEW MAC ADDRESS EACH TIME IT CONNECTS TO A DIFFERENT ACCESS POINT*].	

	The device has been certified to meet the requirements for the Wi-Fi Alliance. The device supports PRF-384 and PRF-704 key generation and TLS v1.2. The device supports mutual authentication using certificates, and can check the validity of the peer certificate according to RFC 5280. If a peer certificate is determined to be invalid the device will not connect to the network and alert the user. The following cipher suites are supported: • TLS_RSA_WITH_AES_256_GCM_SHA384 as defined in RFC 5288, • TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 as defined in RFC 5289, • TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 as defined in RFC 5289, • TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 as defined in RFC 5289, • TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 as defined in RFC 5289	

TS 103 732-2 SFRs

Identification and Authentication

FIA_MBE_EXT.1	The user must be able to enroll their biometric sample to create an authentication template. Document the process for enrolling the user's biometric.	Supported? (Y/N)
	FIA_MBE_EXT.1.1 The TSF shall provide a mechanism to enrol an authenticated user to the biometric system.	
	The device only supports the under-display fingerprint sensor (UDFPS). Users will be asked to touch and hold the fingerprint sensor multiple times (including tips/edges) to complete enrollment. 	***Y***

FIA_MBV_EXT.1	Biometric authentication sensors shall meet minimum requirements for use. Document the FAR/FRR information for each biometric modality available.	Supported? (Y/N)
	FIA_MBV_EXT.1.1 The TSF shall provide a biometric verification mechanism using [fingerprint]. **FIA_MBV_EXT.1.2** The TSF shall provide a biometric verification mechanism with the [FAR] not exceeding [*1:50 000 for fingerprint*] and [FRR] not exceeding [*1:33 for fingerprint*].	***Y***
	USFPS (Ultrasonic Fingerprint Sensor) on the Pixel 10 has met the FAR < 1:50k and FRR < 2.5% thresholds.	

Management

FMT_SMF.1 /BAF	The user must be able to manage their biometric templates. Document what the user can do in terms of managing enrolled biometrics.	Supported? (Y/N)
	FMT_SMF.1.1/BAF The TSF shall be capable of performing the following management functions: [• *ENROL THE INITIAL [*fingerprint*]; AND* • *RE-ENROL OR CHANGE THE [*fingerprint*]*].	***Y***
	Users have the capability to delete their fingerprint enrollments to disable USFPS.	

TS 103 732-4 SFRs

Applications

FAP_LFC.1	The developer shall describe how preloaded apps included in the system image are updated.	Supported? (Y/N)
	The developer can provide a list of the apps along with how they are updated.	
	FAP_LFC.1.1 The TSF shall ensure that preloaded applications are updated by [*using an ADP and system software updates (to the version maintained in firmware)*].	***Y***
	Some preloaded applications are only updated as part of the system software updates but most are provided as apps in the Play Store and can be updated as any application update.	

FAP_LFC.2	The developer shall specify what happens when a preloaded app is uninstalled and the app reverts back to the one included in the current system image.	Supported? (Y/N)
	The developer shall explain the actions that are taken automatically and what the user may be able to do with the app once it has been uninstalled.	
	FAP_LFC.2.1 When a preloaded app update is uninstalled, the TSF shall warn the user the preloaded app will revert to the version in the system image and allow the following actions: [*disable the app, none*].	***Y***
	Some preloaded apps may be able to be disabled (along with uninstalling any app updates). This depends on the app (some are critical to the device and so cannot be disabled). When the app is not able to be disabled the user is warned about uninstalling the updates and then continuing to use the app from that point.	

FAP_LFC.3	The developer shall specify the ADP(s) where preinstalled apps are linked to for updates.	Supported? (Y/N)
	The developer should provide information about the ADP(s) where preinstalled apps will download updates from (updates should not be downloaded from a location that is not an ADP)..	
	FAP_LFC.3.1 The TSF shall ensure that preinstalled applications are only updated from the ADP(s) of the TOE manufacturer and/or OS developer.	***Y***
	All preinstalled apps are installed through the Play Store and so use the Play Store for all updates..	

FAP_LFC.4	The developer shall specify the ADP(s) where preinstalled apps are downloaded from during the installation process.	Supported? (Y/N)
	The developer should provide information about the allowed ADP(s) where apps may be downloaded from. The apps do not have to be specified.	

	FAP_LFC.4.1 The TSF shall ensure that preinstalled applications are only downloaded from the ADP(s) of the TOE manufacturer and/or OS developer.	***Y***
	All preinstalled apps are installed from the Play Store.	

FAP_PRM.1	The developer shall specify the permissions that are restricted to only preloaded and vendor-owned preinstalled apps.	Supported? (Y/N)
	The developer provides a list of system permissions that are restricted to only these apps so that any other app cannot successfully request access to the specified permission.	
	FAP_PRM.1.1 The TSF shall restrict the ability of applications to request [permissions categorized as signature (and can only be assigned to an application that is signed by the platform signing key)] to only preloaded applications and preinstalled applications created by the TOE developer.	***Y***
	Permissions that are categorized at a Protection level of "signature" can only be assigned to apps that are signed by the Google platform signing key. These permissions are not accessible by any apps that are not signed with this key and so cannot be requested. More information about the specific permissions available (and what permissions are categorized as signature) can be found at: https://developer.android.com/reference/android/Manifest.permission	

Application Risk

FPA_RSK.1	No sensitive data should be stored outside of the app container or system credential storage facilities.	Supported? (Y/N)
	Output from a script that checks for proper usage and a report on any apps that could not provide a clear answer from the evaluator.	
	FPA_RSK.1.1 The applications on the TOE shall only store data within their own storage context. **FPA_RSK.1.2** The applications on the TOE shall only store keys using the TSF-provided key storage.	***Y***
	Tested as part of the Preloaded App Scripts	

FPA_RSK.2	The app does not rely on symmetric cryptography with hardcoded keys as a sole method of encryption.	Supported? (Y/N)
	Output from a script that checks for proper usage and a report on any apps that could not provide a clear answer from the evaluator.	

	FPA_RSK.2.1 The applications on the TOE shall use appropriate cryptographic primitives to support the algorithms in use. **FPA_RSK.2.2** The applications on the TOE shall not use deprecated cryptographic modes or algorithms.	***Y***
	Tested as part of the Preloaded App Scripts	

FPA_RSK.3	The app uses cryptographic primitives that are appropriate for the particular use-case, configured with parameters that adhere to industry best practices.	Supported? (Y/N)
	Output from a script that checks for proper usage and a report on any apps that could not provide a clear answer from the evaluator.	
	FPA_RSK.3.1 The applications on the TOE shall not have hardcoded symmetric keys as the method of internal data protection.	***Y***
	Tested as part of the Preloaded App Scripts	

FPA_RSK.4	Data is encrypted on the network using TLS. The secure channel is used consistently throughout the app.	Supported? (Y/N)
	Output from a script that checks for proper usage and a report on any apps that could not provide a clear answer from the evaluator.	
	FPA_RSK.4.1 The applications on the TOE shall not have hardcoded unencrypted URLs for network communications.	***Y***
	Tested as part of the Preloaded App Scripts	

FPA_RSK.5	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.	Supported? (Y/N)
	Output from a script that checks for proper usage and a report on any apps that could not provide a clear answer from the evaluator.	
	FPA_RSK.5.1 The applications on the TOE shall use the trusted channels provided by FTP_ITC_EXT.1/HTTPS or FTP_ITC_EXT.1/TLS.	***Y***
	Tested as part of the Preloaded App Scripts	

FPA_RSK.6	The app verifies the X.509 certificate of the remote endpoint when the secure channel is established.	Supported? (Y/N)
	Output from a script that checks for proper usage and a report on any apps that could not provide a clear answer from the evaluator.	

	FPA_RSK.6.1 The applications on the TOE shall validate the X.509 server certificate according to the following rules: ● The certificate path must terminate with a certificate in the TOE trust anchor; ● The TOE shall use the main OS-provided method to verify the certificate.	***Y***
	Tested as part of the Preloaded App Scripts	

FPA_RSK.7	All inputs from external sources and the user are validated and if necessary sanitized. This includes data received via the UI, IPC mechanisms such as intents, custom URLs, and network sources.	Supported? (Y/N)
	Output from a script that checks for proper usage and a report on any apps that could not provide a clear answer from the evaluator.	
	FPA_RSK.7.1 The applications on the TOE shall sanitize all input from external sources via any available interface.	***Y***
	Tested as part of the Preloaded App Scripts	

FPA_RSK.8	The app does not export sensitive functionality via custom URL schemes, unless these mechanisms are properly protected. Apps by a common app developer may be acceptable in some circumstances.	Supported? (Y/N)
	Output from a script that checks for proper usage and a report on any apps that could not provide a clear answer from the evaluator.	
	FPA_RSK.8.1 The applications on the TOE shall not support unauthorized direct access to data from external apps.	***Y***
	Tested as part of the Preloaded App Scripts	

FPA_RSK.9	The app is signed and provisioned with a valid certificate for the platform.	Supported? (Y/N)
	Output from a script that checks for proper usage and a report on any apps that could not provide a clear answer from the evaluator.	
	FPA_RSK.9.1 The applications on the TOE shall be signed with certificates with a signature format valid for the platform.	***Y***
	Tested as part of the Preloaded App Scripts	

FPA_RSK.10	The app has been built in release mode, with settings appropriate for a release build (e.g. non-debuggable).	Supported? (Y/N)
	Output from a script that checks for proper usage and a report on any apps that could not provide a clear answer from the evaluator.	

	FPA_RSK.10.1 The applications on the TOE shall not have any debug functionality enabled.	***Y***
	Tested as part of the Preloaded App Scripts	

APA_LST.1	Enumerating the preloaded and preinstalled applications with system permissions is necessary to ensure how apps are separate from the OS.	Supported? (Y/N)
	The developer shall list the preloaded apps and any preinstalled apps (ones that are downloaded during the install) that are assigned system permissions.	
	APA_LST.1.1D The developer shall provide a list of all preloaded and preinstalled applications with system permissions included on the consumer mobile device at the completion of the initial CMD setup. **APA_LST.1.1C** The list of preloaded applications shall include all applications contained within the system software. **APA_LST.1.2C** The list of preinstalled applications shall include all applications installed on the consumer mobile device at the completion of the initial CMD setup that have been assigned system permissions. **APA_LST.1.1E** The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.	***Y***
	Google does not specifically distinguish between preloaded and preinstalled apps, all are considered preloaded by internal Google definitions as any application that would have system permissions but is updated through the Google Play Store is considered preloaded (Google does not make any distinction between Preloaded and Preinstalled in the manner specified in the requirements).	

TS 103 732-5 SFRs

Bootloader - User data protection

FDP_ULK.1	If the bootloader is able to be unlocked, user data shall be protected (though wipe).	Supported? (Y/N)
	Describe whether the bootloader can be unlocked, and if so, how the device is wiped when the bootloader state changes. (Locking does not require a wipe, only unlocking)	
	FDP_ULK.1.1 The TSF shall ensure that [*changing the bootloader to an unlocked state will wipe all user data*].	***Y***
	When the bootloader is unlocked the device will be wiped of all user data.	

FDP_BBY.1	Any boot modes must continue to enforce user data security (no bypass)	Supported? (Y/N)
	Describe protections that ensure that alternative boot modes don't bypass the security functions. ***Note this normally would focus on how the data encryption can not be bypassed (i.e. the device booted to user data without any authentication).***	
	FPT_BBY.1.1 The TSF shall be enforced in any alternative boot modes.	***Y***
	There are no boot modes that can bypass the data security.	

Bootloader - Protection of the TSF

FPT_BLP.1	The bootloader must run in the least privileged mode (ARM EL1 for example) possible.	Supported? (Y/N)
	Provide documentation about the boot process and the modes the bootloader runs in, with particular focus on the last stage. If the bootloader is completely removed from memory after loading the OS, then that is sufficient to meet this.	
	FPT_BLP.1.1 The bootloader shall be configured to run in the least privileged mode of the processor.	***Y***
	The bootloader initially loads at EL3 (the first stage of the bootloader), but once it has completed the loading process moves to EL1 to proceed with the boot process. See the boot flow diagram for more detailed information.	

FPT_PRT.1	The partition configuration protection shall be specified to ensure that the system is properly defined.	Supported? (Y/N)
	Provide information about how the integrity of the partition configuration is ensured.	

	FPT_PRT.1.1 The TSF shall protect the integrity of the partition configuration to prevent changes outside of the system image update process.	***Y***
	The integrity of the partition table is maintained using dm-verity.	

FPT_PRT.2	Bootable partitions must be documented, including all settings used on bootable partitions.	Supported? (Y/N)
	Document the partitions of the device. Include both fstab (or equivalent partition table from the device) for all mounted partitions and any additional information for partitions that may not be mounted by the OS but are used for special operations.	
	FPT_PRT.2.1 The TOE has the following partitions marked as bootable: the main OS and [*recovery*]. **FPT_PRT.2.2** The TSF enforces restrictions on writing data, code execution and system permissions through the use of partition flags during the mounting of partitions for use by the TOE.	***Y***
	fstab file for the device Android partition table descriptions	

FPT_ROL.1	Bootloaders must enforce rollback protection for firmware on the device. Tamper-evident storage must support the rollback implementation.	Supported? (Y/N)
	Provide documentation about how rollback protection is implemented and which components support it (and any conditions under which rollback may be bypassed and an earlier version installed).	
	FPT_ROL.1.1 The TSF shall permit rollback of the system software and [*no other software/firmware*] under [*[the case where the bootloader has been unlocked]*] conditions.	***Y***
	System software can only be rolled back when the bootloader has been unlocked (which requires a device wipe first). In the unlocked state any system image may be installed. Pixel followed Android AVB rollback protection mechanism reference: bootloader_source_code.tar.gz	

Bootloader - Functional Specification

ADV_FSP.2	Boot arguments/commands that may be passed to the kernel from the boot process shall be documented.	Supported? (Y/N)
	The focus here is on commands that can be provided outside of the normal programmed commands, so commands from the user that may be passed.	

	This is a modification to the ADV_FSP.2 documentation that is already required as part of the evaluation.	
	As part of the functional specification, the interface between the bootloader and the kernel shall be considered as a TSFI. Boot arguments or commands that may be passed from the bootloader to the kernel outside those that are provided as part of the TOE configuration (such as arguments that may be passed by the user through an external connection to the device) shall be documented and reviewed.	
	Google adds a few internal fastboot oem arguments specific to the device beyond the normal ones found in AOSP. These are: These are related to the boot logos and for manufacturer debugging. There are no other custom arguments in the bootloader. And the commands available do not bypass any security functionality on the device such as enabling a bypass of the authentication process to unlock user data.	***Y***

Root of Trust - Protection of the TSF

FPT_INI.1	The device provides a method for verifying the integrity of the device during the boot process (a Root of Trust).	Supported? (Y/N)
	Document what the Root of Trust is, how it provides support for the initialization and what happens when an error is detected.	
	FPT_INI.1.1 The TOE shall provide an initialization function which is self-protected for integrity and authenticity. **FPT_INI.1.2** The TOE initialization function shall ensure that certain properties hold on certain elements immediately before establishing the TSF in a secure initial state, as specified in FPT_INI.1.2 Table. ID1 [*INTEGRITY*] [*ROOT OF TRUST*] ID2 [*PREVENTION OF DOWNGRADE TO PREVIOUS VERSIONS*] [*ELEMENTS AS SPECIFIED IN FPT_ROL.1.1*] **FPT_INI.1.3** The TOE initialization function shall detect and respond to errors and failures during initialization such that the TOE [*is halted*]. **FPT_INI.1.4** The TOE initialization function shall only interact with the TSF in [during boot time] during initialization.	***Y***
	A public key on the device used to verify the initial bootloader is considered the Root of Trust on the device. This key is fused into the SoC into special OTP eFuses which cannot be changed once they have been set maintaining the integrity of the key.	

	The device utilizes Android Verified Boot 2.0 to ensure the integrity of the system as it loads on the device. The trust of the boot sequence is derived from the public signature key hash which is set into One Time Programmable (OTP) eFuses in the SoC. This hash is used to verify that the signature provided as part of the vmbeta partition is valid (and hence that the partition can be trusted). The vmbeta partition includes information that is used to verify the integrity of the other partitions used to load Android as well as the rollback counter to protect against rollback attacks.	

FPT_RDI.1	The integrity of the stored Root of Trust data is monitored. Explain how the data used by the Root of Trust is monitored for integrity.	Supported? (Y/N)
	FPT_RDI.1.1 The TSF shall monitor Root of Trust data stored in containers controlled by the TSF for integrity errors. A public key on the device used to verify the initial bootloader is considered the Root of Trust on the device. The RoT key is not specifically monitored by a program but is stored into a set of OTP eFuses which cannot be changed once set. Integrity is implied when the bootloader image is successfully verified using the programmed key. If that fails, it will be a problem with the image.	***Y***

GSMA Requirements (FS.56)

Cryptographic Support

FCS_STG_EXT.1	Developers must provide a keystore that is tied to the hardware outside the main OS.	Supported? (Y/N)
	Provide documentation of how the keys are protected in a key store outside the main OS and how this is tied specifically to the hardware.	
	FCS_STG_EXT.1.1 The TSF shall provide [*hardware-based, hardware-isolated*] secure cryptographic key storage. NOTE: Hardware-based => TEE or similar Hardware isolated => eSE, SPU or similar **FCS_STG_EXT.1.2** The TSF shall utilize the provided secure cryptographic key storage for protecting the key hierarchy.	***Y***
	The Pixel 10 has both a hardware-based key store (in Trusty TEE) and hardware-isolated (the Titan M2). These are both used for the Android keystore depending on the specific request of the calling application as to where to store the key (the default is hardware-based).	

Identification and Authentication

FIA_SAR.1	Biometric authentication shall meet minimum requirements to detect spoof attempts.	Supported? (Y/N)
	Document the SAR (or IAPMR) information for each biometric modality available.	
	FIA_SAR.1.1 The TSF shall provide a biometric verification mechanism with the SAR no exceeding [• *below 7%*].	***Y***
	The SAR rate for the UDFPS is 4%.	

Privacy

FPR_ANO.2	The OTA client shall not access user data that is not necessary to perform an update.	Supported? (Y/N)
	Document what data is needed by the OTA client (such as IMEI or email to be authorized to access the update) and how the permissions on the device protect the client from user data. Since the client will have high permissions, showing how the	

	user data is protected from the OTA client (or the client is prevented from accessing the data) is to be shown.	
	FPR_ANO.2.1 The TSF shall ensure that [THE SYSTEM SOFTWARE OTA CLIENT] is unable to determine the real user data bound to [the TOE (HARDWARE PLATFORM)]. **FPR_ANO.2.2** The TSF shall provide [SYSTEM SOFTWARE UPDATES] to [THE USER] without soliciting any reference to the real user data.	
	The OTA client is part of AOSP (https://cs.android.com/android/_/android/platform/system/update_engine) and maintained as part of that project. The OTA client undergoes a privacy review with each Android release. This review (performed by an independent privacy team) ensures that the client does not export PII. Additionally the client is not provided privileges which would provide it access to user/app data.	***Y***

Protection of the TSF

FPT_EAT_EXT.1	Access to telephony commands (such as AT commands or other terminal listeners) via USB or other external interfaces must be disabled at ship time. This is intended to cover commands such as those entered via the dialer to provide various device information and not access to the modem itself (such as programmatic access).	Supported? (Y/N)
	Document how telephony commands can be accessed (both locally through the interfaces and remotely, if possible, from external devices).	
	FPT_EAT_EXT.1.1 The TSF shall only allow access to AT modem commands from [*the user interface*]. **FPT_EAT_EXT.1.2** The TSF shall prompt for approval of AT modem commands sent to the device from outside the user interface [*is not applicable*].	***Y***
	The Pixel 10 does not provide external or remote access to AT commands, they can only be entered directly through the user interface on the device (i.e. the phone dialer app). All access to these commands is disabled as part of the production build process for the telephony system.	

FPT_LNW_EXT.1	Root or system owned processes do not expose any listening network sockets externally or on loopback interfaces. Disabling a listening socket must not require an OTA.	Supported? (Y/N)
	Document the network sockets that are available after the boot has completed from the device. Both loopback and external open ports must be documented.	
	FPT_LNW_EXT.1.1 The TSF shall ensure that no listening network sockets to the external or loopback IP networks are associated with processes with system permissions on the device. **FPT_LNW_EXT.1.2** The TSF shall ensure that [*no network sockets and associated processes*] exposed to the external or loopback IP networks have no system permission on the device.	***Y***
	No network sockets or processes that are exposed to the network have system permissions.	

Life Cycle Requirements

The highlighted text are changes from what is included in the TS 103 732-1 SARs.

ALC_CMC.2	Any third party code shall be added to the developer's CM system.	Supported? (Y/N)
	The process for importing third party code and then maintaining that code shall be described.	
	ALC_CMC.2.4D The developer shall provide guidance for importing and updating external software components into the CM system. **ALC_CMC.2.4C** The CM documentation shall describe the method used to identify external software components and how those components are updated.	***Y***
	All external code is brought into the internal Google git repository	

ALC_CMS.2	Any third party software components shall be documented where applicable.	Supported? (Y/N)
	For external software components, the original maintainer of the software shall be specified.	
	ALC_CMS.2.1C The configuration list shall include the following: the TOE itself; the evaluation evidence required by the SARs; and the parts that comprise the TOE including any external software components. **ALC_CMS.2.2E** The evaluator shall confirm that for external software components, the developer shall include the source and original maintainer of the component.	***Y***

	All external components that are included in the device are tracked in the CM system and included as part of the SBOM for the device. The source of all external components is maintained as part of that inclusion into the internal CM system. Information about how this code is tracked is in the document for ALC_CMC.2.	

ALC_DVS_ EXT.1	The developer shall describe the process for generating the signing keys and unique identifiers on the device (ones that are fixed). The developer needs to provide documentation about how the identifiers are generated and put on the device. This includes how these are secured while in the factory and cleared when not needed. The developer must also describe how signing keys are generated, stored and protected. This includes how they are used (procedures for ensuring rogue builds cannot be created). Here the term keybox is used to identify the location on the device where the unique keys for the device will be stored. This can be in various hardware layers below the OS (the SEE). ***Note that the highlighted sections are where these are changes from the PP. Other sections without changes are not included.***	Supported? (Y/N)
	ALC_DVS_EXT.1.1D The developer shall produce and provide development security documentation on the generation, ~~and~~ protection and use of signing keys~~ and device unique identifiers~~. **ALC_DVS_EXT.1.2D** The developer shall produce and provide development security documentation on the acquisition or generation of device unique identifiers. **ALC_DVS_EXT.1.3D** The developer shall produce and provide development security documentation on the provisioning of data or keys that may be used by a Root of Trust. **ALC_DVS_EXT.1.1C** The development security documentation shall describe all the physical, procedural, personnel, and other security measures that are necessary to protect the confidentiality and integrity of the ~~following manufacturing components:~~ keys used to sign the publicly released system software and its updates. **ALC_DVS_EXT.1.2C** The development security documentation shall describe the procedures for selecting the proper signing keys used for a device and to ensure the use of the proper keys in the build process.	***Y***

	ALC_DVS_EXT.1.3C The development security documentation shall describe all the physical, procedural, personnel, and other security measures that are necessary to protect the confidentiality and integrity of the ~~following manufacturing components:~~ unique, non-modifiable identifiers (such as IMEI, attestation keys or Device Unique Keys) and how they are properly acquired/created and provisioned for each device. **ALC_DVS_EXT.1.4C** The development security documentation shall describe all the physical, procedural, personnel, and other security measures that are necessary to protect the confidentiality and integrity of the ~~following manufacturing components:~~ data and keys provisioned to the device for a Root of Trust for the device.	
	Pixel devices are manufactured in facilities that are ISO 27001 & 9001 certified, following best practices for security measures in manufacturing. Some keys and identifiers are provisioned through a system built by Google to securely pass the data from Google to the device directly during manufacturing. An active/online server called Pulsar is used to directly connect, while a backup server called Janus is installed locally at the factory which can provide keys when the main connection is offline. The Janus server is a caching server meant to hold a certain amount of data to enable devices to be provisioned even when connectivity is inconsistent. Data from the provisioning system is sent to both Pulsar and Janus at the same time to keep this data in sync regardless of which server is used (the preference is the Pulsar system). The provisioning system itself generates the packages that will be deployed to a device. These packages are generated on the provisioning system using the chip_id and an ECDSA public key provided by the SoC manufacturer. The ID and the private key are stored in OTP FUSEs in the chip. This combines the IMEIs, MACs and keybox that are to be delivered to the device into a single encrypted package (encrypted using the ECDSA key) that can only be read by that particular device. The encrypted packages are delivered in batches to Janus, but are retrieved live with calls from Pulsar. The ECDSA public key is considered the Root of Trust for the device as that key verifies the initial code to be loaded on the system. This process starts when the device is plugged into a provisioning station where the chip_id is read. This is passed to Pulsar and then to the system backend (or to Janus). If the chip_id matches one that is expected, the generated package	

	will be delivered to the device. The encrypted package is decrypted on-device inside the TEE and written to storage (encrypted with the locally generated root key). IMEIs are acquired from GSMA and MACs are acquired from IEEE. The keybox and other device keys are generated through a service provided for all Android OEMs. The root encryption keys (for the device and for the Titan M2 chip) are generated on-device using their respective RNGs. These keys never leave the device (or in the case of the Titan chip, never leave the chip). These keys are considered the Root of Trust for the device. Attestation keys are derived from these keys as needed. Google manages signing keys through an internal service called Remote Keymaster. This is based on the Keymaster library (used by many Google systems). The primary difference in Remote Keymaster (RKM) is where the key materials are stored. In the case of the RKM, the keys are stored on centrally managed, highly locked-down servers. These servers are designed to provide high assurance of confidentiality and integrity, but not availability (not set in the Google Cloud). The RKM servers are access-controlled, limiting the number of people with physical access in a Google datacenter (which is already limited access). RKM servers have additional restrictions on personnel who have physical access to the servers, and all access is logged (badge access). RKM servers are not accessible from the internet. The Keymaster library (see the original design doc here) utilizes standard cryptography for all operations (AES, SHA, HMAC, RSA, ECDSA) for its operations. In the RKM implementation, an interface is provided to the user to perform actions remotely and none of the key material is ever retrieved from the server to the local client. So while the Keymaster on its own is designed for local use, when contained within RKM, it provides for a secure, remote keystore. The RKM servers log every use of a private key, so every action for signing is audited based on the Google ID of the person (or system) initiating the action. Alerts based on usage are provided (to provide immediate notification instead of requiring the audit record to be reviewed constantly).	

	There are three classes of build keys associated with the types of builds which can be generated. Two of these (AOSP builds and Dev builds) have their keys stored within the codebase inside the Google build systems. These keys are not related to the production keys and as such are considered to be self-managed keys. These keys are not accepted on production hardware as they are signed with a different signing chain to prevent them being used for anything other than development purposes on devices that are used for testing and engineering development. The public key hash stored in the Pixel device does not correspond to these keys. The third class of build keys are the production keys that are generated in RKM. The production signing keys never leave the RKM server and all actions taken to sign the builds occur on the RKM server. The artifact generated in the build process to create the signature is provided to the RKM where it is signed and returned. This process is handled specifically by the employees tasked with this signing, and the number of users with this access is very restricted by policy (and as above, audited). This process is used for both full images and OTA images. The Google build process to ensure that authorized and untampered-with software is called Binary and Configuration ID (BCID). Android builds follow the L3 requirements in the build process. Individual SKUs for a Pixel device are each assigned specific keys so binaries cannot be used across different devices, even of the same model. As access to the signing keys is tightly controlled and not accessible in the clear (even on disk), it is highly unlikely for a signing key to become public. The RKM does support rotation of keys though, so it is possible for a key to be changed within the build system and the key management system.	

ALC_FLR.3	The developer shall have a process for receiving information about flaws and then provide patches for those flaws to the impacted devices. In addition this includes a defined period and frequency of updates to the device (including both OS updates and regular security patches).	Supported? (Y/N)
	The developer needs to provide documentation about the flaw remediation/patching process. This can include public sites/information along with the support information for the device under evaluation.	

	Note that the highlighted sections are where these are changes from the PP.	
	ALC_FLR.3.1D The developer shall document and provide flaw remediation procedures addressed to TOE manufacturers. **ALC_FLR.3.2D** The developer shall establish a procedure for accepting and acting upon all reports of security flaws and requests for corrections to those flaws. **ALC_FLR.3.3D** The developer shall provide flaw remediation guidance addressed to TOE users. **ALC_FLR.3.4D** The developer shall provide public guidance related to the duration period, frequency and type of updates that will be released to support the TOE. **ALC_FLR.3.5D** The developer shall provide a public vulnerability disclosure program to provide security bulletins about the flaws that have been remediated. **ALC_FLR.3.6D** The developer shall establish procedures for ensuring that no known security vulnerabilities rated as High and Critical (e.g. as classified in public databases) are included in the TOE at public release. **ALC_FLR.3.1C** The flaw remediation procedures documentation shall describe the procedures used to track all reported security flaws in each release of the TOE. **ALC_FLR.3.2C** The flaw remediation procedures shall require that a description of the nature and effect of each security flaw be provided, as well as the status of finding a correction to that flaw. **ALC_FLR.3.3C** The flaw remediation procedures shall require that corrective actions be identified for each of the security flaws. **ALC_FLR.3.4C** The flaw remediation procedures documentation shall describe the methods used to provide flaw information, corrections and guidance on corrective actions to TOE users. ~~The flaw remediation procedures documentation shall also define the planned minimum length of time after release of the TOE that these methods will be used to maintain the TOE.~~ **ALC_FLR.3.5C** The flaw remediation procedures shall describe a means by which the developer receives from TOE users reports and enquiries of suspected security flaws in the TOE. **ALC_FLR.3.6C** The flaw remediation procedures shall include a procedure requiring timely response and the automatic	***Y***

	distribution of security flaw reports and the associated corrections to registered users who might be affected by the security flaw. **ALC_FLR.3.7C** The procedures for processing reported security flaws shall ensure that any reported flaws are remediated and the remediation procedures issued to TOE users. **ALC_FLR.3.8C** The procedures for processing reported security flaws shall provide safeguards that any corrections to these security flaws do not introduce any new flaws. **ALC_FLR.3.9C** The flaw remediation guidance shall describe a means by which TOE users report to the developer any suspected security flaws in the TOE. **ALC_FLR.3.10C** The flaw remediation guidance shall describe a means by which TOE users may register with the developer, to be eligible to receive security flaw reports and corrections. **ALC_FLR.3.11C** The flaw remediation guidance shall identify the specific points of contact for all reports and enquiries about security issues involving the TOE. **ALC_FLR.3.12C** The flaw remediation procedures documentation shall define the planned minimum duration after release of the TOE that these methods will be used to maintain the TOE. **ALC_FLR.3.13C** The flaw remediation procedures documentation shall define the types of updates (such as security/maintenance or operating system) and the frequency of these updates being provided for the TOE. **ALC_FLR.3.14C** The flaw remediation procedures documentation shall describe the process for publicly releasing security flaw remediation information, including the location(s) where this will be publicly available. **ALC_FLR.3.15C** The flaw remediation procedures documentation shall describe the process for verifying known security flaws are not propagated into a new (not yet released) TOE. **ALC_FLR.3.1E** The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.	
	The device has a defined lifecycle as can be seen here with Android OS updates and monthly security patches provided until May 2031.	

	Google supports a bug filing system for the Android OS outlined here: https://source.android.com/setup/contribute/report-bugs. This allows developers or users to search for, file, and vote on bugs that need to be fixed. This helps to ensure that all bugs that affect large numbers of people get pushed up in priority to be fixed. The method outlined above requires the user to submit their bug to Android's website. As such, the user of the device needs to establish a trusted channel web connection to securely file the bug by following the set-up steps to establish a secure HTTPS/TLS connection from the TOE, then visiting the above web portal to submit the report. Google posts Android Security Bulletins every month about the patches that have been released with each monthly patch. Monthly patches are distributed automatically to the devices. As the release placed on a device at launch is an iteration of the Android release cycle (generally this would be a new version of Android, such as 12 to 13), the process of patching the OS is ongoing and continued as part of the release process. When a new device is released, all patches up to that date will be included with the device. This will be the same as for a device that will receive that OS release as an OTA update. Google partners with component vendors to ensure support for the chipsets, including patches related to security updates for the duration of the support period for the device. In this case, the Pixel 10 guarantees component support until at least May 2031 (as some chips may be used on newer devices, support may be longer for some components). The contracts with the suppliers require patches to be produced based on the threat level of the vulnerability following the same threat levels used for all Android bug reports (suppliers must meet the expected timelines for providing patches). Google works in partnership with creating patches and testing, and then rolls these out to Android. Note that Google provides patches for components that may not be used by Google but by partner OEMS as well. All software used by Google is maintained inside an internal git repository. Google has added many layers to incorporate review cycles and commenting on top of git. All actions taken in the repository (and all the layers on top) are tied to Google corporate accounts, and access controls ensure that engineers can only make changes to code for which they have responsibility. While other engineers may suggest changes, these must be approved by the engineers specifically responsible for the software to be merged in. Google actively tracks external repositories on which it relies for code in both Android and its own apps. Tracking is automated	

	wherever possible with code updates being automatically copied for review into the internal repositories (it is not automatically applied until after a standard review as for any code change). Once the code is copied into the Google repository, it is managed like any other internally generated code. The Google VDP information can be found here: https://bughunters.google.com/about/rules/6171833274204160/android-and-google-devices-security-reward-program-rules Every update download from a device includes metadata about the device sent back to Google servers. This provides for both tracking of the numbers of devices that have specific patch levels installed as well as success rates for installation. Successful completion of the update will trigger a notice back to the server of success. Google requires a security review before the release of any new device. This encompasses the hardware and software of the device based on a standard review that was initially employed with the Pixel 2 device series. This IR plan is updated and used as the basis for all new devices. The security review process is a blocking gate in the release of any product, and cannot be bypassed.	