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
X500 Pro Max	Android	17	6.18

Device	Model(s)	CPU Architecture	CPU
X500 Pro Max	V2610/J2603	ARM64	MT6995

Equivalent Devices

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

Claimed Device	Evaluated Device	Differences between devices
X500 Pro Max, V2610/J2603	X500 Pro, V2621/V2622	HW design

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.

FCS_RNG_EXT.1	Devices shall provide at least one random number generator (RNG) that produces uniformly-distributed, unpredictable output.	Supported? (Y/N)
	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.	
	FCS_RNG_EXT.1.1 The TSF shall provide a [*deterministic*] random number generator. **FCS_RNG_EXT.1.2** The TSF shall provide random numbers that meet [NIST SP 800-90B].	*Y*
	The device (running the Android system) implement a deterministic random bit generator compliant with the NIST SP 800-90B standard via BoringSSL or the Java Cryptography Architecture (JCA), typically either CTR_DRBG with AES-256 or HMAC_DRBG with SHA-256. This standard fully meets the qualification criteria for new cryptographic algorithms specified in Annex A of ISO/IEC 18033-1.	

FCS_CKM.1/Asymmetric	Devices shall generate asymmetric keys properly that can meet at least the equivalent of 128-bit symmetric encryption strength.	Supported? (Y/N)
	The process for generating asymmetric keys on the device shall be explained. This will be specific to the algorithm(s) in use.	
	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	Y

	cryptographic key sizes [RSA keys of 3072, 4096, ECDSA keys of P-256, P-384, P-521] that meet the following: [FIPS 186-5].	
	The device utilizes both the BoringSSL library and the hardware Trusted Execution Environment (TEE) to generate RSA and ECDSA keys. Both implementations strictly comply with NIST FIPS 186-5 and ensure a minimum security strength equivalent to 128-bit symmetric keys (e.g., RSA 3072 and ECC P-256 or higher). Furthermore, all key generation processes rely on the DRBG specified in FCS_RNG_EXT.1 to provide the necessary randomness.	

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 ~~[~~*FOR SYMMETRIC ENCRYPTION USING A RANDOM NUMBER GENERATOR AS SPECIFIED IN FCS_RNG_EXT.1, and FOR SYMMETRIC ENCRYPTION A COMBINATION OF PRECURSOR KEY MATERIAL USING A DERIVATION FUNCTION AS SPECIFIED IN FCS_COP.1/DERIVATION*] and specified cryptographic key sizes [*256 bits*] ~~that meet the following: [**assignment**: *list of standards*~~].	Y
	For symmetric key generation, the device utilizes both the hardware Trusted Execution Environment (TEE) and the BoringSSL module. When generating a new symmetric key (such as an AES-256 key), the TSF directly invokes the DRBG specified in FCS_RNG_EXT.1 to acquire high-entropy random bytes. Alternatively, when deriving keys from precursor material (e.g., shared secrets or user credentials), the TSF employs standard Key Derivation Functions (such as HKDF or PBKDF2) as specified in FCS_COP.1/DERIVATION. Both methods ensure the generated symmetric keys possess a security strength of at least 128 bits, strictly complying with NIST SP 800-133 and related standards.	

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 and ECDSA*] and cryptographic key sizes [RSA keys of 3072, 4096, ECDSA keys of P-256, P-384, P-521] that meet the following: [FIPS 186-5].	Y
	FIPS 186-5 support for both RSA and ECDSA.	

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 SP 800-56A Revision 3*].	Y

	To secure user data in transit, the device supports robust key establishment protocols primarily through the BoringSSL cryptographic module. During the establishment of secure communication channels (such as TLS 1.2/1.3 or IPsec), the TSF utilizes the Elliptic Curve Diffie-Hellman (ECDH) algorithm for key agreement. By employing NIST standard curves (P-256, P-384, and P-521), the key establishment process guarantees a security strength of 128 bits or higher, which perfectly matches or exceeds the strength of the symmetric encryption keys (e.g., AES-128 or AES-256) subsequently used for bulk data encryption. This implementation strictly adheres to the NIST SP 800-56A standard.	

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 (SP800-38F), GCM (SP800-38D), XTS (SP800-38E) and GCM (SP800-38D) modes].	Y
	To protect user data both at rest and in transit, the device employs the Advanced Encryption Standard (AES) as its primary symmetric encryption algorithm. All implementations strictly utilize key sizes of 128 bits or 256 bits, complying with FIPS 197 and relevant NIST/IEEE operational mode standards, ensuring robust security for user data.	

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-based Extract-and-Expand Key Derivation*	Y

	Function (HKDF) and Password-Based Key Derivation Function 2 (PBKDF2)] and cryptographic key sizes [*128 bits, 256 bits*] that meet the following: [*RFC 5869 (for HKDF) and NIST SP 800-132 (for PBKDF2)*].	
	The device supports robust Key Derivation Functions (KDFs) to securely generate cryptographic keys from precursor materials or user credentials, utilizing both the hardware TEE and the BoringSSL module. For deriving cryptographic keys from user-supplied authentication data (e.g., lock screen PIN or password), the device employs PBKDF2, incorporating a secure salt and a high iteration count. For system-level key derivation, such as generating sub-keys for File-Based Encryption (FBE) or deriving session keys during TLS 1.3 handshakes, the device utilizes HKDF (HMAC-based KDF). Both functions are configured to output key sizes of 128 bits or 256 bits, which perfectly match the key sizes used for subsequent AES encryption. These implementations strictly adhere to NIST SP 800-132 and RFC 5869 standards, respectively.	

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 [*SHA-256, SHA-384, and SHA-512*] and cryptographic key sizes [*NONE*] that meet the following: [*FIPS 180-4*].	Y
	The device utilizes cryptographic hashing for various critical security functions, including data integrity verification, digital signature generation and verification (for OTA updates and APK installations), and certificate validation. To perform these operations, the device's Trusted Execution Environment (TEE) and the software-based BoringSSL cryptographic module implement the SHA-2 family of hash algorithms, specifically SHA-256, SHA-384, and SHA-512. As cryptographic hash functions are unkeyed, no key sizes are applicable. All implemented hash algorithms strictly comply with the Secure Hash Standard defined in FIPS 180-4.	

	Devices shall support a message authentication code algorithm.	

FCS_COP.1 /KeyedHash	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).	Supported? (Y/N)
	FCS_COP.1.1/KeyedHash The TSF shall perform [*KEYED-HASH MESSAGE AUTHENTICATION*] in accordance with a specified cryptographic algorithm [*HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512*] and cryptographic key sizes [*256 bits, 384 bits, and 512 bits*] that meet the following: [*FIPS 198-1*].	Y
	To ensure data integrity and data origin authentication, the device supports Keyed-Hash Message Authentication Code (HMAC) algorithms. These are extensively used in secure network communications (e.g., TLS/IPsec), Android Keystore operations, and the validation of Hardware Authentication Tokens within the Trusted Execution Environment (TEE). The device implements HMAC in conjunction with the SHA-2 hash family, specifically supporting HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512. The cryptographic key sizes used for these operations are 256 bits, 384 bits, and 512 bits, respectively, matching the output size of the underlying hash functions to ensure maximum security. All HMAC implementations strictly comply with the FIPS 198-1 standard.	

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*].	Y

	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*].	
	The device supports Device Encrypted (DE) storage to protect Low User Data Assets, allowing critical applications (like alarms and phone calls) to function during Direct Boot before the user has authenticated. https://source.android.com/docs/security/features/encryption/file-based https://developer.android.com/training/articles/direct-boot.html	

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*].	Y

	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*].	
	MEDIUM (Credential Encrypted Storage - CE) https://source.android.com/docs/security/features/encryption/file-based https://developer.android.com/training/articles/direct-boot.html The devices implement Android's File-Based Encryption (FBE), providing a robust key hierarchy for Credential Encrypted (CE) storage to protect Medium User Data Assets. By default, all user data and third-party application data are stored in the CE storage area. The CE key hierarchy cryptographically binds the master data encryption keys to both the user's authentication credential (PIN, pattern, or password) and the device's hardware root key (Device Unique Key / DUK). Consequently, CE data remains fully encrypted and inaccessible until the user successfully authenticates to unlock the device. **HIGH (Hardware-Backed Keystore & App-Specific Encryption)** For High User Data Assets, the devices provide advanced capabilities allowing applications to implement strict, state-aware encryption utilizing the hardware-backed Android Keystore (isolated within the vivo TEE). Applications can generate specialized encryption keys with rigid access control policies via Keystore APIs, specifically: 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 an application utilizes these hardware-bound Keystore keys to individually encrypt its own sensitive files (application-layer encryption), it guarantees that High User Data can only be decrypted and accessed when the authorized user is actively present and the device is unlocked. This requires specific implementation by the app developer to utilize these high-security APIs on top of the default CE storage.	

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 VOLATILE MEMORY, BY A SINGLE DIRECT OVERWRITE CONSISTING OF [A RANDOM PATTERN];* • *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 (Target of Evaluation) clears sensitive cryptographic material (plaintext keys, authentication data, and other security parameters) from memory when no longer needed. Volatile Memory (RAM): Public keys can remain in memory when the phone is locked, but all crypto-related private/secret keys are strictly evicted from memory upon device lock or when the cryptographic operation concludes. Non-Volatile Memory (Flash): No plaintext cryptographic material resides in the TOE's Flash, as the TOE encrypts all keys stored in Flash.	

	When performing a wipe of protected data (e.g., factory reset or user deletion), the TOE utilizes Cryptographic Erasure. It effectively erases the protected data by clearing the Data-At-Rest Data Encryption Key (DEK). Because the Android Keystore resides within the user data partition, clearing the Data-At-Rest DEK cryptographically erases all associated Keystore keys simultaneously. To physically destroy these top-level keys (Data-At-Rest DEK and Secure Key Storage SEK), the TOE executes a secure direct overwrite using the Linux BLKSECDISCARD ioctl command on the wear-leveled Flash memory containing the key, followed by a read-verify operation to ensure complete destruction.	

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*	Y

	ACTIONS RELATED TO THE UPDATE_APP OPERATION OR ACCESS TO THE APP DURING ITS UPDATING]. **FDP_UPF_EXT.1.1/APP_Update** The TSF shall be able to check for a [APP] update package every [***once a day***].	
	The device utilizes the Android Package Manager Service (PMS) and the pre-installed application store (e.g., vivo App Store / Google Play Store) to manage app updates securely. Frequency of automatic checking: The application store background service is configured to automatically check for available app updates at least once a day (typically when the device is connected to Wi-Fi and charging). Method for verifying new updates (Versioning): The Android PMS extracts the versionCode (an integer value) from the manifest of the new APK. It strictly compares this against the versionCode of the currently installed app. If the new versionCode is lower than the installed one, the PMS rejects the installation with an INSTALL_FAILED_VERSION_DOWNGRADE error, effectively preventing downgrade attacks. Method for verifying validity (Signature checks): Before applying an update, the PMS rigorously verifies the cryptographic signature of the new APK using the Android APK Signature Scheme (v2/v3/v4). It ensures that the certificate used to sign the update package is identical to the certificate of the currently installed application. If the signatures do not match, the update is rejected (INSTALL_FAILED_UPDATE_INCOMPATIBLE). Atomic Update & Access Denial: The installation process is atomic. During the actual replacement of the application files, the OS forcefully terminates any running processes of the target app and temporarily blocks access to it until the atomic installation transaction is successfully completed or rolled back.	

FDP_ACC.2 /SSW_Update **FDP_ACF.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)	

FDP_UPF_EXT.1/SSW_Update	• 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: [*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:* ○ *[selection: 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*].	Y

	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 system software updates using a public key chaining to the vivo root public key. The SHA-256 hash of this root public key is permanently fused into the SoC (System-on-Chip) hardware during manufacturing, establishing a hardware root of trust. Once an OTA update has been downloaded, the version and signature are strictly checked by the system to ensure the version is not older than the currently installed version (Anti-rollback/Anti-downgrade protection) and that the cryptographic signature is valid (verified against the hardware-fused hash). If the checks complete successfully, the update process begins in the background. The device utilizes the Android A/B (Seamless) System Updates architecture (see: https://source.android.com/docs/core/ota/ab). The update is installed onto the slot (partition) that is not currently in use (i.e., the inactive slot that was not used to boot the device). Once the update has been completely written to the inactive slot, the update process will set the updated slot as active and prompt the user to restart the device. If the subsequent reboot into the new slot is not successful (e.g., due to corruption or tampering), the bootloader will automatically switch back to the previous slot and restart again, returning the device to its previous secure state. If the signature checks, write process, or reboot fail at any time during this sequence, the system will remain or revert to the OS version that existed prior to the update. This ensures the update process happens completely or not at all (maintaining strict atomicity). The device checks for updates on a daily basis (once every 24 hours) by communicating with the vivo update servers. When an update is found, the user is notified to download and install the update, or they can schedule it to be installed at a later time.	

	Devices shall provide access controls (permissions) to components on the system and provide the user with the	Supported? (Y/N)

FDP_ACC.2 /Permissions **FDP_ACF.1/ Permissions**	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.	
	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];* ● *MANUFACTURER OBJECTS: DEVICE ID, SYSTEM PERMISSIONS, FILES (INCLUDING INDIVIDUAL APP DATA), [SECURE SOCKETS, SECURE 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*	Y

	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 device implements a robust, multi-layered access control system based on the Android permission model and SELinux. ● List of user permissions: These are "Dangerous/Runtime Permissions" that handle sensitive user data. The list includes, but is not limited to: Camera, Microphone, Location, Contacts, Calendar, Call Logs, SMS/Text Messages, Stored Pictures/Media, List of Installed Apps, Bluetooth (Nearby Devices), and Body Sensors. ● List of OS permissions: These are "Normal/Signature Permissions" and system-level access controls managed by the OS. The list includes: Device ID (IMEI/MAC), System Settings, Internet/Network Access, Secure IPC (Binder communication), Secure Sockets, and access to individual app data directories (sandboxing). ● How permissions are granted: ➢ User Permissions: Granted explicitly by the user via runtime dialog prompts (Allow/Deny) when a third-party app first attempts to access the resource. ➢ OS/Pre-installed Permissions: "Normal" permissions are granted automatically at install time. For pre-installed system applications, permissions are granted automatically without user interaction based on their location in the read-only system partition and their cryptographic signature (matching the platform signature). ● How access is determined (allow/block): When an app (subject) requests access to a resource (object), the Android	

	framework checks the application's assigned UID (User ID) and its associated granted permission list. If the permission is explicitly granted (by user or system), access is allowed. If the user explicitly blocked it, or if the rule granting access does not exist, the framework denies the request (throwing a SecurityException). ● SELinux as the basis: Underpinning the framework-level permissions is SELinux (Security-Enhanced Linux), which operates in 'Enforcing' mode. SELinux provides Mandatory Access Control (MAC). Every process, file, socket, and IPC node is assigned an SELinux security context. Even if an app bypasses the framework permission checks, the kernel-level SELinux policy will strictly block the access if the app's process context does not have explicit allow rules to interact with the target object's context.	

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*	Y

	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 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 using Android File-Based Encryption (FBE). By default, all user data is protected with a key that is derived from the user's lock screen credential (PIN/password/biometric). This is considered Medium protection (Credential Encrypted - CE storage). The device only supports internal storage. For Low data (Device Encrypted - DE storage), an application must be explicitly developed to take advantage of this feature. This allows specific data saved by the app to be available on reboot before the user has authenticated (e.g., for alarms or receiving calls in Direct Boot mode). 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 (specifically the Android Keystore system using the UNLOCKED_DEVICE_REQUIRED flag) provide the ability to hook into the notification of the lock status to enable this functionality. When the screen locks, the keys protecting this data become unavailable. In all cases, keys that are stored on the device are entangled with the hardware root key backed by the SoC's Trusted Execution Environment (TEE) or Secure Element (SE). In the case of Low data, 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 specific physical device where the data was 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 [● Enter password to unlock ● Make/receive emergency calls ● Take pictures (stored internally) - unless the camera was disabled ● Turn the TOE off (Power off) ● Restart the TOE (Reboot) ● See notifications (note that some notifications identify actions, for example to view a screenshot; however, selecting those notifications highlights the password prompt and requires the password to access that data) ● Configure sound, vibrate, or mute ● Set the volume (up and down) for ringtone ● Enable and disable Flashlight(When the screen is off, press and hold the volume button to turn on, and click the power button to turn off) ● Take Screenshots(Three-finger sliding screenshot)] 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 [*list of TSF mediated actions, which are listed in FIA_UAU.1.1*] 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.	Y
	The device enforces strict access control before user identification and authentication are successfully completed. Prior to a successful login, access to Credential Encrypted (CE) user data is strictly prohibited. However, to maintain basic functionality and safety, the TSF allows a specific set of actions to be performed on behalf of the	

	user before they are authenticated. These actions do not allow access to previously stored sensitive user data.	

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, 2D FACE, FINGERPRINT*] to support user authentication. **FIA_UAU.5.2/Local** The TSF shall authenticate any user's claimed identity according to the [*The user must establish a primary authentication credential (PIN, Password, or Pattern) before configuring any secondary biometric authentication (Fingerprint or Face). When multiple methods are enabled, the device accepts either the primary credential or a valid secondary biometric input to unlock, except in specific fallback scenarios (e.g., after reboot, timeout, or biometric failure) where only the primary credential is accepted.*].	Y
	The device supports multiple local authentication methods to verify the user's claimed identity. **Supported Authentication Methods**: 1. Primary Credentials (Knowledge-based): PIN, Password, and Pattern. 2. Secondary Credentials (Biometric): Fingerprint recognition and 2D Face recognition. **Ordering and Rules for Multiple Authentication Methods**: The device employs a hierarchical authentication model managed by the Android Keystore and Gatekeeper subsystems: 1. Prerequisite: A user cannot configure a biometric method (Fingerprint or Face) without first establishing a primary credential (PIN, Password, or Pattern). The primary credential acts as the cryptographic root for device encryption (CE data). 2. Concurrent Usage (Normal Operation): If both a primary credential and biometric methods are enabled, the lock	

	screen simultaneously prompts for both. The user can choose to authenticate using either the fingerprint sensor, facial recognition, or by swiping up to enter their PIN/Password/Pattern. The first successful method will unlock the device. 3. Fallback/Enforcement Rules (Primary Credential Required): The system enforces specific security policies where secondary biometric methods are temporarily disabled, and the device strictly requires the primary credential (PIN/Password/Pattern). This occurs in the following scenarios: ➢ First Unlock after Power-on/Reboot: Biometrics cannot decrypt CE storage; the primary credential is required. ➢ Timeout: The device has not been unlocked using the primary credential for a specific period . ➢ Idle: The device has been idle/locked for a prolonged period since the last biometric unlock. ➢ Failed Attempts: After too many consecutive failed biometric attempts, biometrics are locked out. ➢ Lockdown Mode: If the user manually triggers the Android 'Lockdown' feature, biometrics are disabled until the next primary credential entry. ➢ Device Administration Policy: If an MDM/Enterprise policy strictly mandates PIN/Password entry.	

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, PINS, USING A COMMON USER ACCOUNT TO A REMOTE SERVICE ON BOTH DEVICES*] **for**	Y

	~~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*].	
	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 Access Point (AP) or establish a Wi-Fi Direct peer connection. ● Note that QR codes can be read automatically by the camera, but by default are handed to the web browser or specific system handlers and are not implicitly tied to establishing a trusted device-level connection without user intent. ● NFC labels: These can be used to provide out-of-band (OOB) connectivity and authentication information for pairing to Bluetooth devices or initiating peer-to-peer file sharing (e.g., Android Beam / Nearby Share). ● Accounts on the device: Apps and system services can register accounts (seen in Settings -> Passwords & accounts) that can be used to connect to remote services (such as a mail server, backup/sync server, or other cloud services). These require the user to enter a valid username/password combination on the device that matches an account on the remote 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).	Supported? (Y/N)
	Document what actions require authentication: ● Unlocking the device after it has become locked (mandatory) ● Change of authentication data (any change) (mandatory) ● Other conditions?	
	FIA_UAU.6.1 The TSF shall re-authenticate the user under the conditions [	Y

	• *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 & privacy -> Device unlock menu in the user interface. Only after entering their current user password can the user then elect to change their password or biometric data.	

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.	Y
	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/face unlock attempt succeeded or failed.	

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 [	Y

	• *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.	

FIA_AFL.1	When repeated authentication attempts are detected, the device should deter continued attempts. This includes all available authentication methods.	Supported? (Y/N)
	Document what actions are taken when some number of attempts have been detected (between 3-10).	
	FIA_AFL.1.1 The TSF shall detect when [*AN INTEGER BETWEEN 3 AND 10*] unsuccessful authentication attempts occur related to [*PASSWORD, PIN, PATTERN, **2D FACE, FINGERPRINT***]. **FIA_AFL.1.2** When the defined number of unsuccessful authentication attempts has been [MET], the TSF shall [*REBOOT, PROGRESSIVELY LENGTHEN THE TIME TO ATTEMPT AN AUTHENTICATION, MAKE ALL USER DATA ASSETS UNREADABLE, [Block the biometric use after 5 unsuccessful attempts. Require a 30-second delay after 5 unsuccessful non-biometric attempts, with the delay progressively increasing for subsequent failures.]*].	Y
	The device maintains in persistent storage the number of failed authentication attempts since the last login. **Biometric Authentication**: Biometric authentication (fingerprint or face) can be attempted up to 5 consecutive times. After 5 failed attempts, biometric authentication is disabled. The user must then use their primary	

	credential (password, PIN, or pattern) to unlock the device and re-enable biometrics. **Non-Biometric Authentication (Password, PIN, Pattern)**: The device implements a progressive delay mechanism to deter brute-force attacks: After 5 consecutive failed attempts, the device forces the user to wait for 30 seconds before further attempts are allowed. For additional failed attempts beyond 5, the device progressively lengthens the mandatory waiting period (e.g., increasing to minutes or longer) to severely limit the rate of guessing.	

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.	Supported? (Y/N)
	Document what the user can do in setting permissions for access (approve/reject persistently or temporarily, etc)	
	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.	Y
	The user can manage the permissions for an application based on defined permission groups (such as Camera, Location, Microphone, Contacts, etc.). Applications that are preloaded (or that are part of the system) are assigned necessary permissions by the system to ensure the core functionality of the device, but are not assigned more permissions than are necessary for the function of the app (principle of least privilege).	

	Apps that are installed by the user are assigned a restrictive default (no sensitive permissions are granted during installation). The permissions that are needed by the app will be requested at runtime on first use. At that time, the user is prompted and can choose to: Approve persistently: Allow continued access (e.g., "While using the app" or "Allow"). Approve temporarily: Allow access only for that specific session (e.g., "Only this time"). Reject persistently/temporarily: Deny access to the permission (e.g., "Don't allow"). If rejected, the app may prompt again in the future unless the user selects a "Don't ask again" equivalent. Permissions for an app can be modified at any time after they have initially been set through the device settings: Settings -> Apps -> App manager -> <app in list> -> Permissions Here the user can choose to modify (approve or reject) the permissions that have been previously granted or denied.	

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 unlock From here the user can choose Screen lock, Fingerprint or Face to change their credentials (set, remove, modify, etc). Setting a fingerprint and 2D face will require the user to also set a password, PIN or pattern.	

	The user shall be able to manage the permissions provided to apps and some system services	Supported? (Y/N)

FMT_SMF.1 /Permissions	• 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*].	
	The user can view, grant, and revoke permissions for any installed application through the device Settings menu. Users can manage these permissions using two primary methods: 1. By Application: The user can navigate to Settings -> Apps -> App manager -> <app in list> -> Permissions Here, the user can view all the permissions requested by that specific app. The user can select individual permissions (such as Camera, Location, or Storage/Files) to grant or revoke them (e.g., changing from "Allow" to "Don't allow"). 2. By Permission Type (Permission Manager): The user can navigate to Settings -> Apps -> App manager -> <app in list> -> Permissions This interface allows the user to view permissions grouped by category. For example, the user can select "Files and media" to see exactly which apps have been granted read and/or write access to user objects. From this list, the user can select an app to grant or revoke that specific access.	Y

FMT_SMF.1 /UserControls	The user shall be able to manage various settings on the device • 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: [	Y

	• *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 grant or revoke permission for apps to access the accessibility service through: Settings -> Shortcuts & accessibility -> Accessibility The user can manage device notifications (including which apps can send notifications, access to read notifications, and whether they can be seen on the lock screen) through: Settings -> Notifications and status bar The user can manage USB connectivity when the device is connected to another device (such as a PC) via a wired charging interface. By default, the connection is set to "Charge only" (No data transfer) to protect user data. When connected, a prompt or notification appears, allowing the user to access USB preferences and manually change the mode to "File transfer / USB tethering" or "Photo transfer (PTP)". The device does not support removable media (MicroSD cards).	

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: [~~assignment:~~ selection: • *CHANGE OR RESET THE PRIVACY ALIASES;* • *BLOCK THE CREATION/USE OF A UNIQUE ID FOR ADVERTISING (NO PERSONALIZED TRACKING)*].	Y

	The user can access the advertising controls and manage privacy aliases (Advertising ID) in: Settings -> Security & privacy -> More privacy settings -> Ads Here, the user can choose to "Reset advertising ID" (which replaces the current identifier with a new random one) or "Delete advertising ID" completely. If the advertising ID is deleted, apps will receive a string of zeros instead of an identifier, effectively blocking the creation/use of a unique ID for personalized advertising tracking. Regarding app-specific identifiers (aliases generated by individual apps), the user generally does not have direct access to view them as they are managed internally by the app. However, if the user wants to reset or delete an app-specific identifier, they can do so by clearing the app's storage data: Settings -> Apps -> App manager -> <app in list> -> Storage -> Clear data This action will delete all local identifiers and associated app data, ensuring that a new identifier is created the next time the app is launched.	

FMT_SMF.1/APP_Update	The user shall be able to manage applications on the device (update/uninstall) Describe how application updates can be initiated and how apps (including pre-installed) can be uninstalled.	Supported? (Y/N)
	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

	The user can manage application updates and installations primarily through the pre-installed application distribution platform (ADP), such as the Google Play Store (or vivo V-Appstore). Application Updates: Within the ADP settings (e.g., Play Store -> Profile icon -> Settings -> Network preferences -> Auto-update apps), the user can specify the update behavior: Automatically install: Apps will update automatically over Wi-Fi (or any network, based on user choice). Notify without downloading: The user can turn off auto-updates, in which case the ADP will periodically check for updates and notify the user that updates are available, requiring manual initiation. The user can initiate an immediate check for updates by opening the ADP and navigating to the "Manage apps & device" section. From there, the user can initiate an update for a specific app or update all available apps. Displaying Version Number and Uninstalling: The user can view detailed information about any installed app, including its version number, and uninstall it through the device Settings: Settings -> Apps -> App manager -> <app in list> Version Number: The app version is displayed at the bottom of the App info page (or directly under the app name). Uninstall: The user can uninstall downloaded apps by tapping the "Uninstall" button on this same App info page. Alternatively, users can long-press the app icon on the home screen and select "Uninstall".	

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*	Y

	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*].	
	The user can check for new system software updates, view the current system version, and manage update settings by navigating to: Settings -> System update **Update Management**: Check and Initiate: Opening the "System update" page initiates an immediate check for updates. If an update is available, the user can manually initiate the download and installation. Specify Behavior: By tapping the settings icon (gear icon) on the top right of the System update page, the user can configure "Smart Upgrade" (or Auto-download over Wi-Fi). Users can choose to have updates automatically downloaded and installed (typically overnight), or disable this to only be notified without downloading. Delay Installation: If an update is downloaded and ready, the user is prompted to install it. The user can choose to install immediately or delay it temporarily (e.g., select "Install overnight" or dismiss the prompt to install later). **Status and Version**: Status: The "System update" page displays the real-time status and progress of the update download and installation process. Upon successful reboot, a notification confirms the update result. Version Number: The current system software version number is prominently displayed on the Settings -> System update page, and also in Settings -> About phone.	

Privacy

FPR_PSE.1	Advertisers and app developers shall be provided a unique device identifier that can be modified by the user. Describe how the unique identifier is provided and how this can be changed by the user.	Supported? (Y/N)
	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 and third-party apps are not able to access unique hardware identifiers due to the strict access control permissions enforced by the OS (target API level restrictions prevent access to non-resettable device identifiers). The user can manage and reset this advertiser identifier string in: Settings -> Security & privacy -> More privacy settings -> Ads For App Developers, the system allows unique identifiers to be generated to enable identification of the device/user within their specific services. Similar to the advertising ID, this 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 their apps, but this is an internal developer choice). An app developer can use standard Android APIs, such as java.util.UUID.randomUUID(), to request and generate a random identifier for their application's use.	

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
	The device provides multiple hardware-based storage locations for keys to maintain confidentiality and integrity. The DUK 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 device also provides support for the StrongBox Keystore. 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.	

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 using File-Based Encryption (FBE). Except for Device Encrypted (DE) data which is necessary for basic boot, Credential Encrypted (CE) user data cannot be decrypted without a successful user authentication (PIN/Password/Biometrics). Therefore, a failure to update the system software cannot unlock or expose user data, as the compromised or failing new system would not be able to properly start and prompt for, or process, the user's credentials. Furthermore, the device utilizes A/B (Seamless) System Updates. The device maintains two boot slots (Slot A and Slot B) for the system. Only one slot is active at a time, and the alternate, inactive slot is used to install the update in the background. If the update process fails at any point (such as an interruption during the write operation, or if upon reboot the new system slot cannot successfully boot and reach a state where the user can authenticate), the system's bootloader will automatically mark the new slot as unbootable. The system will then automatically revert to the original, known-good boot slot and return to the previous version of Android (the version running prior to the update attempt), preserving the secure state and all user data.	

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*	Y

	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 [selection: SEE, *[executable code stored in /system and /vendor partitions]*] BY INTEGRITY, WHERE INTEGRITY IS VERIFIED BY [*AN IMMUTABLE HARDWARE 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*].	
	The device runs a number of self-tests on specific components to ensure they are properly functioning during the start-up process. Failure of these tests will cause the boot sequence to halt and a Boot Failure message will be shown to the user. The device verifies the integrity of the following system components during start-up (Android Verified Boot): 1. Bootloader 2. OS kernel 3. TEE (Trusted Execution Environment) 4. Executable code stored in /system and /vendor partitions All these components are verified by checking their digital signatures using a chain of trust rooted in the hash of an asymmetric key that is permanently fused into the SoC hardware (eFuses) during manufacturing.	

FPT_RCV.2	The device shall support a maintenance mode to enable recovery from malicious/persistent software.	
	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)	Supported? (Y/N)
	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	Y

	ensure the return of the TOE to a secure state using automated procedures.	
	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), minor errors will be automatically corrected by the dm-verity check during start-up using Forward Error Correction (FEC). In these cases, the error is detected and corrected automatically without user intervention. For larger failures that cannot be automatically corrected, or if the bootloader/kernel integrity checks fail (e.g., persistent malicious modifications to core components), the device will halt the normal boot process to prevent the execution of compromised code. The device will then boot into a maintenance state (Recovery Mode or Fastboot Mode). In this secure state, the user or administrator can take manual actions to restore the system: 1. Perform a "Wipe data/factory reset" to remove potentially malicious user-installed applications and data. 2. Manually reload known-good, officially signed firmware onto the device. Users can apply official OTA update packages via the Recovery interface, or utilize official vivo service tools and firmware packages provided by vivo authorized support channels to restore the device to a secure factory state.	

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 [V6.0]*] 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 6.0 specification. Pairing is not allowed without explicit authorization from the user (this authorization cannot be programmatically bypassed or entered, but must be input directly from the user via the system UI). If the device is already paired and connected with a peer, a second peer attempting to connect with the same BD_ADDR (Bluetooth Device Address) will be rejected. This ensures that a	

	malicious device spoofing the MAC address of the first peer will be blocked from connecting while the original legitimate device is already connected. To ensure cryptographic forward secrecy, each time the device establishes a connection to a peer over Bluetooth, a new ECDH (Elliptic-Curve Diffie-Hellman) public/private key pair will be generated specifically for that session.	

FTP_ITC_EXT.1/HTTPS **FTP_ITC_EXT.1/TLS**	The device shall support TLS 1.2 or higher	Supported? (Y/N)
	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)	
	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.	Y

	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*]. **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 [selection: NOT ESTABLISH THE CONNECTION, REQUEST APPLICATION AUTHORIZATION TO ESTABLISH THE CONNECTION, NO OTHER ACTION] IF THE PEER CERTIFICATE IS DEEMED INVALID; AND* ● *SUPPORTS THE FOLLOWING CIPHER SUITES [* ○ *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_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,* ○ *TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (IETF RFC 5289),* ○ *TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 (IETF RFC 7905),* ○ *TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (IETF RFC 7905)*].	
	The device supports mutual authentication using certificates, and can check the validity of the peer certificate according to RFC 5280. The underlying Android operating system supports the ability for an app to check the certificate revocation status using OCSP. The app must implement the relevant APIs to enable the check and determine the action to take if the certificate is found to be	

	revoked (the OS does not perform any action based on the revocation information itself other than notifying the app requesting the check of the state). The device acts as a client and 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.	

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*]. **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], TLS V1.3 [10]]; AND* ● *SUPPORTS THE FOLLOWING CIPHER SUITES [* ○ *TLS_RSA_WITH_AES_256_GCM_SHA384 (IETF RFC 5288 [8]),* ○ *TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 (IETF RFC 5289 [9]),*	Y

	○ *TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (IETF RFC 5289 [9]),* ○ *TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 (IETF RFC 5289 [9]),* ○ *TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (IETF RFC 5289 [9]),* ○ *TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 (IETF RFC 5289 [9]),* ○ *TLS_ECDHE_ECDSA_WITH_AES_256_CBC_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 supports PRF-384 and PRF-704 key generation and utilizes TLS v1.2 and TLS v1.3 for secure WLAN communications. The device supports mutual authentication using certificates (such as EAP-TLS), 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 reject the certificate and will not establish the connection. Furthermore, the device supports the required TLS cipher suites for WLAN authentication and randomly generates a new MAC address each time it connects to a different access point.	

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.	Y
	The vivo device supports biometric authentication using fingerprint and facial recognition. 1. Fingerprint Enrollment: The user can enroll a fingerprint by repeatedly pressing the fingerprint sensor from different angles until the biometric template is fully captured. 2. Face Enrollment: The user can enroll their face by registering their facial features using the front-facing camera. The enrollment process can be accessed via the following path: Settings -> Security & privacy -> Device unlock -> Fingerprint / Face	

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, 2D FACE*]. **FIA_MBV_EXT.1.2** The TSF shall provide a biometric verification mechanism with the [FAR] not exceeding [*1:50 000 FOR 2D FACE, 1:50 000 FOR FINGERPRINT*] and [FRR] not exceeding [*1:20 FOR 2D FACE, 1:33 FOR FINGERPRINT*].	Y
	The device supports both fingerprint and 2D facial recognition. The biometric sensors and underlying algorithms have been tested and verified to ensure that the False Acceptance Rate (FAR) and False Rejection Rate (FRR) for both modalities strictly meet or exceed the specified security 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 [2D FACE, FINGERPRINT]; AND* • *RE-ENROL OR CHANGE THE [2D FACE, FINGERPRINT]*].	Y
	User could delete, disable their enrolled fingerprint or 2D face. And add their fingerprint.	

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
	For the device, the update mechanism for preloaded applications depends on the specific app type: *1. System Software Updates: Core system applications tightly integrated into the OS are updated exclusively through Over-The-Air (OTA) system firmware updates provided by vivo.* *2. Authorized Distribution Platform (ADP): Many preloaded applications (such as vivo system apps or Google Mobile Services apps) can be updated independently via an ADP, such as the Google Play Store or the official vivo App Store. When updated via an ADP, the new version resides in the user data partition, while the original version is maintained in the read-only firmware.*	

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*].	Y
	On devices, when a user attempts to uninstall an update for a preloaded application, the system displays a warning prompt notifying the user that the application will be replaced with the factory version and all associated app data will be removed. After the update is successfully uninstalled and the app reverts to the version maintained in the system image, the system interface allows the user the option to "Disable" the application if they do not wish to use it.	

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
	For devices, preinstalled applications are exclusively linked to and updated from authorized distribution platforms (ADPs). Specifically, updates are securely downloaded either from the Google Play Store (the OS developer's ADP) or the official vivo App Store / V-Appstore (the TOE manufacturer's ADP), depending on the specific application.	

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
	During the installation or provisioning process, any preinstalled applications that require downloading are strictly sourced from authorized distribution platforms (ADPs). The allowed ADPs for devices are the Google Play Store or the official vivo App Store (V-Appstore).	

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 [p*ermissions 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 vivo 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
	vivo provides a comprehensive list of all preloaded and preinstalled applications that are assigned system permissions upon completion of the initial device setup. This detailed list is	

	submitted to the evaluator as a separate proprietary document to satisfy APA_LST.1.1D, APA_LST.1.1C, and APA_LST.1.2C. *For evaluation and verification purposes (APA_LST.1.1E), the evaluator can manually inspect the granted permissions of any specific preloaded application directly on the device by navigating to:* Settings -> Apps -> App manager -> <app in list> -> Permissions	

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 [*the bootloader cannot be unlocked*].	Y
	The production device's bootloader cannot be unlocked.	

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 boot process follows a secure chain of trust. The initial stages of the bootloader execute at the highest privilege level (EL3) to establish the secure environment. Once the primary loading and verification are complete, the bootloader drops privileges and transitions to EL1 (Non-Secure Supervisor mode) to execute the main Android Bootloader (ABL/LK) and proceed with loading the OS.	

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
	The standard AOSP bootable partitions follow the standard Android partition architecture. A complete list of all partitions, including vivo-specific customized partitions and the device fstab detailing mount flags, is provided to the evaluator in a separate proprietary document.	

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 is in an unlocked state. However, the bootloader on production vivo devices is securely locked, thereby strictly enforcing rollback protection. vivo implements the standard Android Verified Boot (AVB) rollback protection mechanism, utilizing secure, tamper-evident hardware storage to store the rollback index, preventing the installation of older, potentially vulnerable firmware versions.	

Bootloader - Functional Specification

ADV_FSP.2	Boot arguments/commands that may be passed to the kernel from the boot process shall be documented. 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.	Supported? (Y/N)
	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.	Y
	Similar to standard Android implementations, vivo utilizes specific fastboot oem (or proprietary) commands for internal manufacturing, diagnostics, and authorized servicing, in addition to standard AOSP commands. However, on production devices where the bootloader is locked, the execution of sensitive commands is strictly restricted. Most importantly, none of the commands available to an end-user can be used to alter the kernel command line in a malicious way, nor do they bypass any security functionality on the device . A detailed list of TSFI boot arguments is available for evaluator review in the supplementary design documentation.	

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). Document what the Root of Trust is, how it provides support for the initialization and what happens when an error is detected.	Supported? (Y/N)
	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*]	Y

	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.	
	The Hardware Root of Trust (RoT) on vivo devices is established by a public OEM key hash that is cryptographically fused into the System-on-Chip (SoC) using One-Time Programmable (OTP) eFuses during manufacturing. Once blown, these eFuses cannot be altered, ensuring the physical integrity of the RoT. *During the boot process, the device utilizes this hardware RoT in conjunction with Android Verified Boot (AVB) to establish a secure chain of trust. The hardware-fused key hash verifies the primary bootloader, which in turn verifies the signature of the vbmeta partition. The vbmeta partition contains the cryptographic descriptors and rollback counters required to verify the integrity and authenticity of all subsequent OS partitions (e.g., boot, system, vendor) before they are loaded.* *If any integrity check fails at any stage of the boot sequence (e.g., signature mismatch or rollback attempt detected), the boot process is strictly halted to prevent the execution of compromised or unauthorized code.*	

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.	Y
	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.*	

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*] 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 device utilizes a hardware-based key store implemented within the Trusted Execution Environment (TEE). The TEE operates in a secure world entirely isolated from the main Android OS (Rich Execution Environment). The Android Keystore system routes cryptographic key generation and storage requests to the Keymaster/KeyMint Trusted Application (TA) running inside the TEE, ensuring that raw key material is never exposed to the main OS.	

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 [selection: • *BELOW 7%*].	Y
	The SAR is lower than 7%.	

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
	On production devices (user builds), external or remote access to the modem via AT commands over physical interfaces (such as USB diagnostics ports) or wireless interfaces is strictly disabled at ship time.	
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 and third-party code (including AOSP, vendor BSPs, and open-source libraries) is ingested into vivo's internal, access-controlled Git-based Configuration Management (CM) system. The process for importing, tracking, and updating these external components is strictly defined in vivo's internal software development lifecycle (SDLC) documentation. This ensures all external code is version-controlled and subjected to the same security scanning and build processes as proprietary code. c	

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.	Y

	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.	
	All external and third-party software components integrated into the device are strictly tracked within vivo's internal Configuration Management (CM) system and are comprehensively documented in the device's Software Bill of Materials (SBOM). The configuration list and SBOM explicitly record the source repository, version, and the original maintainer/author for each external component. See attached ALC_CMS.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).	Supported? (Y/N)
	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.***	
	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.	Y

	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. **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.	
	vivo has fully complied with this requirement by formulating and strictly implementing security documents for development and manufacturing such as the Key Security Management System and Factory Production Flashing Process. In the system development and release phase, vivo adopts Hardware Security Modules (HSMs) together with strict physical access control, dual-person authorization and permission isolation mechanisms to protect private keys for system signing, and ensures that automated compilation pipelines can accurately match keys for corresponding device models. In the device manufacturing phase, vivo has built a controlled secure factory flashing environment. Through encrypted links and dedicated production line serial number writing tools, device unique identifiers including IMEI, as well as Root of Trust (RoT) data such as Secure Boot, are securely and accurately injected into the hardware eFuses or secure storage of each device. Comprehensive physical, procedural and personnel security control measures are implemented throughout all the above processes, effectively safeguarding the confidentiality and integrity of core production data.	

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.	Y

	ALC_FLR.3.6C The flaw remediation procedures shall include a procedure requiring timely response and the automatic 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.	
	As described in document "ALC_FLR.3_Review"	