Honeywell Mobility Edge Android 9 Administrator Guidance Documentation

Version 1.2
2021/03/02

1. Document Introduction

This guide includes procedures for configuring NIAP Common Criteria on a Honeywell Mobility Edge device.

1.1 Evaluated Devices

The evaluated devices are in the table below, all sharing the common Honeywell Mobility Edge platform with the Android 9 operating system.

Product	Model #	CPU	Kernel	Android OS version	Security Patch Level	WFA #
CN80G	CN80-L1N	SDM660	4.4.153	Android 9.0	January 2021	WFA102378
CN80G	CN80-L0N	SDM660	4.4.153	Android 9.0	January 2021	WFA102377
CK65	CK65-L0N	SDM660	4.4.153	Android 9.0	January 2021	WFA102376
CT60	CT60-L0N	SDM660	4.4.153	Android 9.0	January 2021	WFA90429
CT60	CT60-L1N	SDM660	4.4.153	Android 9.0	January 2021	WFA83421
CT40	CT40P-L0N	SDM660	4.4.153	Android 9.0	January 2021	WFA100511
CT40	CT40P-L1N	SDM660	4.4.153	Android 9.0	January 2021	WFA100512

Each of the hardware models included in this evaluation has the ability to run either an AOSP (Android Open-Source Project) or a GMS (Google Mobile Services) version of the Android 9 operating system. The AOSP version is a purely open-source based version of Android that does not contain any Google Mobile Services, which are a collection of Google applications and APIs. The GMS version of software contains Google's applications (such as Chrome, Gmail, Google Maps, etc) along with an additional set of APIs. There is no special configuration required for a particular hardware model to support GMS or AOSP and images can be interchanged on the same device without any special provisioning. Both versions of the Operating System are included in this evaluation.

To verify the build number on your device:

1. Tap on HUpgrader
2. The device will inform the user of the latest image version installed.

1.2 Acronyms

- AE – Android Enterprise
- AES – Advanced Encryption Standard
- API – Application Programming Interface
- CA – Certificate Authority
- DO – Device Owner
- DPC – Device Policy Controller
- EMM – Enterprise Mobility Management
- MDM – Mobile Device Management
- PKI – Public Key Infrastructure
- TOE – Target of Evaluation

2. Evaluated Capabilities

The NIAP Common Criteria configuration adds support for several security capabilities which may be required for your organization. Some of these capabilities include the following:

- Data Protection
- Screen Lock
- Key Management
- Device Integrity
- Device Management
- Work Profile Separation
- VPN Connectivity
- Audit Logging

2.1 Data Protection

Honeywell Mobility edge devices use industry-leading security features to protect user data using FIPS validated security modules for both data-in-transit as well as data-at-rest. The platform creates an application environment that protects the confidentiality, integrity, and availability of user data.

2.1.1 Full-disk Encryption

Encryption is the process of encoding user data on an Android device using an encryption key. With encryption, even if an unauthorized party tries to access the data, they won't be able to read it. The TOE utilizes Full-Disk encryption (FDE) which allows the entire partition to be encrypted in case the device is compromised.

The TOE also provides users with the ability to protect Data-At-Rest with AES encryption, including all user and mobile application data stored in the user's data partition. The TOE uses a key hierarchy that combines a REK with the user's password to provide protection to all user and application cryptographic keys stored in the TOE. The TOE is architected in such a way that a single operating system image can be loaded to each of the different device configurations (CN80G, CT60, CT40, CK65 etc.) This architecture is designed in such a way as to allow runtime configuration of the system to accommodate different display, keyboard, scan engine, wireless connectivity etc. based on the SKU of the device. The SKU configuration of the device is stored in a non-volatile EEPROM outside the application processor. This configuration is loaded during manufacturing.

Direct Boot allows encrypted devices to boot straight to the lock screen and allows alarms to operate, accessibility services to be available and phones to receive calls before a user has provided their credentials.

2.2 Lock screen

Honeywell Mobility Edge NIAP validated government devices use passwords as the primary authentication during the lock screen. Password complexity can be defined through an MDM policy. The STIG guide can be used for instructions on how to configure password complexity requirements. A common-access-card (CAC) can be used after a user authenticates through the Lock Screen dependent on the necessary supporting applications being installed.

Android's GateKeeper throttling is also used to prevent brute-force attacks. After a user enters an incorrect password, GateKeeper APIs return a value in milliseconds in which the caller must wait before attempting to validate another password. Any attempts before the defined amount of time has passed will be ignored by GateKeeper. Gatekeeper also keeps a count of the number of failed validation attempts since the last successful attempt. These two values together are used to prevent brute-force attacks of the TOE's password.

2.3 Key Management

2.3.1 KeyStore

The Android KeyStore class lets you manage private keys in secure hardware to make them more difficult to extract from the device. The KeyStore enables apps to generate and store credentials used for authentication, encryption, or signing purposes.

Keystore supports symmetric cryptographic primitives such as AES (Advanced Encryption Standard) and HMAC (Keyed-Hash Message Authentication Code) and asymmetric cryptographic algorithms such as RSA and EC. Access controls are specified during key generation and enforced for the lifetime of the key. Keys can be restricted to be usable only after the user has authenticated, and only for specified purposes or with specified cryptographic parameters. For more information, see the Authorization Tags and Functions pages.

Additionally, version binding binds keys to an operating system and patch level version. This ensures that an attacker who discovers a weakness in an old version of system or TEE software cannot roll a device back to the vulnerable version and use keys created with the newer version.

On Honeywell Mobility Edge devices, the KeyStore is implemented in secure hardware. This guarantees that even in the event of a kernel compromise, KeyStore keys are not extractable from the secure hardware.

2.4 KeyChain

The KeyChain class allows apps to use the system credential storage for private keys and certificate chains. KeyChain is often used by applications and system services including the Chrome/Chromium browser, Virtual Private Network (VPN) apps, and many enterprise apps to access keys imported by the user or by the mobile device management app.

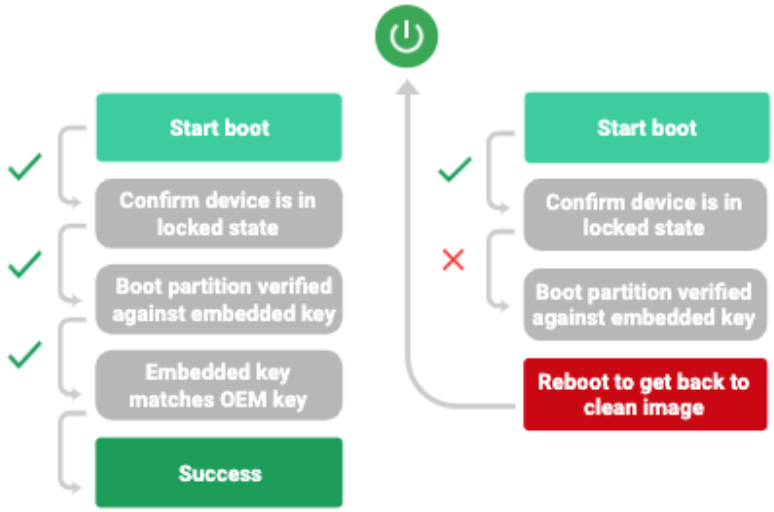
Whereas the KeyStore is for non-shareable app-specific keys, KeyChain is for keys that are meant to be shared across profiles. For example, your mobile device management agent can import a key that the web browser application will use for an enterprise website.

2.5 Device Integrity

Device integrity features protect Honeywell Mobility Edge devices from running an operating system image that has been compromised. With key business and workflow operations using mobile devices for essential communication and core productivity tasks, keeping operating system integrity in place is mandatory. Without operating system integrity, essential security controls can be bypassed (such as access to the filesystem, application segmentation, app armor, etc.). Android adopts several measures to guarantee device integrity at all times, including verified boot, kernel sec comp protection, Android Keystore and Keymaster, etc.

2.5.1 Verified Boot

Verified Boot is Android's secure boot process that verifies system software before running it. This makes it more difficult for software attacks to persist across reboot as well as to ensure there are no persistent malicious applications or software installed, and provides users with a safe state at boot time. Each Verified Boot stage is cryptographically signed. Each phase of the boot process verifies the integrity of the subsequent phase, prior to executing that code. Full boot of a compatible device with a locked bootloader proceeds only if the OS satisfies integrity checks. Verification algorithms used must be as strong as current recommendations from NIST for hashing algorithms (SHA-256) and public key sizes (RSA-2048).



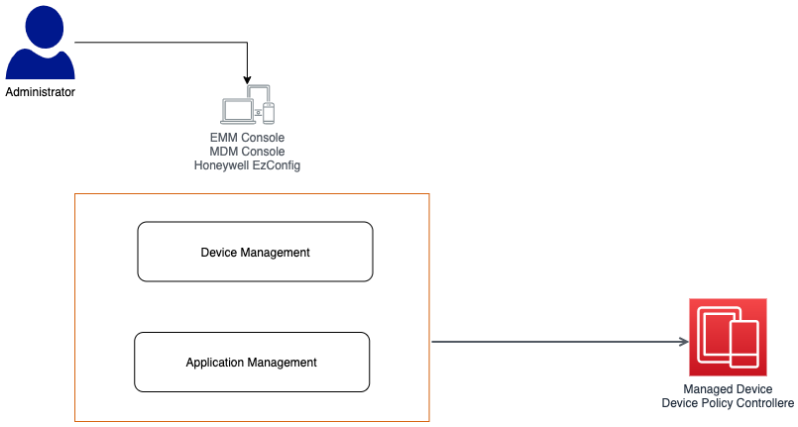
The Verified Boot state is used as an input in the process to derive disk encryption keys. If the Verified Boot state changes (e.g. the user unlocks the bootloader), secure hardware prevents access to data used to derive the disk encryption keys that were used when the bootloader was locked. Verified Boot on compatible devices require rollback protection. This means that in case of an event such as an OS compromise or physical attack an attacker cannot put an older, vulnerable, firmware on the device and boot it. Additionally, rollback protection state is also stored in tamper-evident storage.

Organizations can check the state of Verified Boot using KeyStore key attestation. This retrieves a statement signed by the secure hardware attesting to many attributes of Verified Boot along with other information about the state of the device.

Find out more about Verified Boot here.

2.6 Device Management

The TOE leverages the device management capabilities that are provided through Android Enterprise which is a combination of three components: your EMM/MDM console, a device policy controller (DPC) which is your MDM Agent, and an EMM/MDM Application Catalog.



Components of an Android Enterprise solution.

2.6.1 EMM/MDM console

EMM solutions typically take the form of an EMM console—a web application that allows IT admins to manage their organization, devices, and apps. To support these functions for Android, you integrate your console with the APIs and UI components provided by Android Enterprise.

2.6.2 DPC (MDM Agent)

All Android devices that an organization manages through your EMM console must install a DPC app during setup. A DPC is an agent that applies the management policies set in your EMM console to devices. Depending on which development option you choose, you can couple your EMM solution with Android's DPC or with a custom DPC.

Organizations can provision a fully managed or dedicated device using a DPC identifier (e.g. "afw#"), according to the implementation guidelines defined in the Play EMM API developer documentation.

- The EMM's DPC must be publicly available on Google Play, and the end user must be able to install the DPC from the device setup wizard by entering a DPC-specific identifier.
- Once installed, the EMM's DPC must guide the user through the process of provisioning a fully managed or dedicated device.

2.7 Work Profile Separation

Fully managed devices with work profiles are for company-owned devices that are used for both work and personal purposes. The organization still manages the entire device. However, the separation of work data and apps into a work profile allows organizations to enforce two separate sets of policies. For example:

- A stronger set of policies for the work profile that applies to all work apps and data
- A more lightweight set of policies for the personal profile that applies to the user's personal apps and data

For organizations that are using Honeywell Mobility Edge devices as a "single purpose" solution, where the device is only used in support of a specific business workflow, usually there is no need for managing separate business and personal profiles.

You can learn more about work profile separation in section 7.

2.8 VPN Connectivity

Device administrators can configure an "Always ON" VPN to ensure that data all data or all data from a managed profile will always go through the configured VPN tunnel. **Note:** this feature requires deploying a VPN client that supports the "Always ON" feature. Device administrators can specify an arbitrary VPN package to be set as an Always On VPN. Administrators can use managed configurations to specify the VPN settings for an application.

You can read more about VPN configuration options in section 6.

2.9 Audit Logging

Device Administrators can gather usage data from devices that can be parsed and evaluated for malicious for anomalous and potentially malicious behavior. Activities logged include (among others) Android Debug Bridge (adb) activity, application launch, and screen unlocks:

- Device Administrators can enable security logging for target devices, and the EMM's DPC must be able to retrieve both security logs and pre-reboot security logs automatically.
- Device Administrators can review enterprise security logs for a given device and configurable time window, in the EMMs console.
- Device Administrators can export enterprise security logs from the EMMs console.

Device Administrators can also capture relevant logging information by using Intents as well as Logcat functions which do not require any additional configuration to be enabled.

You can see a detailed audit logging table in section 9.

3. Security Configuration

Honeywell Mobility Edge devices offer a rich built-in interface and an MDM interface for granular security configuration. This section identifies the security parameters for configuring your device in Common Criteria mode and for managing its security settings.

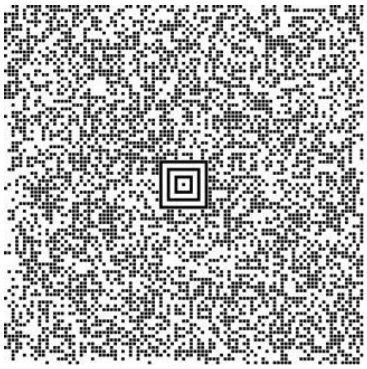
3.1 Common Criteria Mode

To configure the device into Common Criteria Mode, you must set the following options:

1. Require a lock screen password
   - Please review the Password Management items in section 3.4 (Common Criteria Related Settings)
2. Disable Smart Lock
   - Smart Lock can be disabled using KEYGUARD_DISABLE_TRUST_AGENTS()
3. Disable Debugging Features (Developer options)
   - By default, debugging features are disabled. The system administrator can prevent the user from enabling them by using DISALLOW_DEBUGGING_FEATURES()

4. Disable installation of applications from unknown sources
   - This can be disabled by using DISALLOW_INSTALL_UNKNOWN_SOURCES()
5. VPN Full Tunnel Configuration
   - In order to leverage full tunnel IPSEC VPN, the VPN client must be configured to route all traffic (0.0.0.0) through the VPN application
6. Enable Audit Logging
   - Audit Logging can be enabled using setSecurityLoggingEnabled.
   - For certain items, Intents and Logcat can be used which don't require any additional enablement

Honeywell Mobility Edge devices can also be configured for NIAP Common Criteria mode by simply scanning the QR Code below, after "Provisioning Mode" has been enabled. In order to enable Provisioning Mode please refer to the user guide for your specific device.



For instructions on how to generate your own barcode to enable NIAP CC mode, please see below.

Honeywell Mobility Edge devices have several alternate methods in which to enable:
Common Criteria Mode via your MDM console:

1. Ask the MDM Administrator to edit the following item in DeviceConfig.xml
   Modify item: DeviceConfig--->Other settings -->MDM Settings --> Enable_NIAP_Mode
   Value sample: 1: Enable NIAP mode; 0: Disable NIAP mode
2. In MDM Administration Console, MDM Administrator will package this DeviceConfig.xml and push this package to Honeywell Mobility Edge device

Common Criteria via Honeywell EZConfig:

1. Enable Provisioning Mode via:
   Settings -- Honeywell Settings -- Provisioning Mode -- Turned On
2. Enter Power Tools -- EZConfig -- Generator -- Open DeviceConfig.xml and edit the following item in DeviceConfig.xml:
   Modify item: DeviceConfig--->Other settings -->MDM Settings --> Enable_NIAP_Mode
   Value sample: 1: Enable NIAP mode; 0: Disable NIAP mode
3. Open menu in EZConfig application, select **Save** then select **Update Configure**

Common Criteria via a scan of a barcode:

1. Import DeviceConfig.xml into Enterprise Provisioner tool and edit the following item in DeviceConfig.xml
   Modify item: DeviceConfig--->Other settings -->MDM Settings --> Enable_NIAP_Mode
   Value sample: 1: Enable NIAP mode; 0: Disable NIAP mode
2. Select **generate barcode** in Enterprise Provisioner tool
3. Scan the barcode

No additional configuration is required to ensure key generation, key sizes, hash sizes, and all other cryptographic functions meet NIAP requirements.

3.2 Cryptographic Module Identification

The TOE implements CAVP certified cryptographic algorithms which are provided by the following cryptographic components:

1. BoringSSL Library:
   - BoringCrypto version 2.0
2. Android LockSettings service KBKDF
   - Version 77561fc30db9aedc1f50f5b07504aa65b4268b88
3. Hardware Cryptography:
   - TOE's Wi-Fi Chipset provides an AES-CCMP implementation
   - The TOE's application processor (Snapdragon 660 [SDM660]) provides additional cryptographic algorithms. The CAVP certificates correctly identify the specific hardware.

The use of other cryptographic components beyond those listed above was neither evaluated nor tested during the TOE's Common Criteria evaluation.

No additional configuration is needed in order for the cryptographic modules to be compliant.

3.3 Permissions Model

Android runs all apps inside sandboxes to prevent malicious or misbehaving application code from compromising or interfering with other apps or the operating system. Because application sandboxing is enforced at the kernel, this enforcement extends to the entire app regardless of the specific development environment, APIs used, or programming language. A memory corruption error in an application only allows arbitrary code execution in the context of that application, with the permissions enforced by the OS.

Similarly, system components run in least-privileged sandboxes in order to prevent compromises in one component from affecting others. For example, externally reachable components, like the media server and WebView, are isolated in their own restricted sandbox.

Android employs several sandboxing techniques including Security-Enhanced Linux (SELinux), seccomp, app armor, and file-system permissions.

The purpose of a *permission* is to protect the privacy and security of an Android user and their data. Android apps must request permission to access sensitive user data (such as contacts and SMS), as

well as certain system features (such as camera and internet). Depending on the feature, the system might grant the permission automatically or might prompt the user to approve the request.

A central design point of the Android security architecture is that no app, by default, has permission to perform any operations that would adversely impact other apps, the operating system, or the user. This includes reading or writing the user's private data (such as contacts or emails), reading or writing another app's files, performing network access, keeping the device awake, and so on.

The DPC can pre-grant or pre-deny specific permissions using PERMISSION_GRANT_STATE API's. In addition, the end user can revoke a specific app's permission by:

1. Tapping on Settings>Apps and notifications
2. Tapping on the particular app and then tapping on Permissions
3. From there the user can toggle off any specific permission

You can learn more about Android Permissions on developer.android.com.

3.4 Common Criteria Related Settings

The Common Criteria evaluation requires a range of security settings be available. Those security settings are identified in the table below. In many cases, the administrator or user must have the ability to configure the setting, but no specific value is required.

Security Feature	Setting	Description	Required Value	API	User Interface
Encryption	Device Encryption	Encrypts all internal storage	N/A	Encryption on by default with no way to turn off	
	Wipe Device	Removes all data from device	No required value	wipeData()	To wipe the device, go to Settings>System>Reset options and select **Erase all data (factory reset)**
	Wipe Enterprise Data	Remove all enterprise data from device	No required value	wipeData() called from secondary user	
Password Management	Password Length	Minimum number of characters in a password	No required value	setPasswordMinimumLength()	To set a screen lock go to Settings>Security & location>Screen lock and tap on **Password**
	Password Complexity	Specify the type of characters required in a password	No required value	setPasswordQuality()	To set a screen lock go to Settings>Security & location>Screen lock and tap on **Password**
	Password Expiration	Maximum length of time before a password must change	No required value	setPasswordExpirationTimeout()	
	Authentication Failures	Maximum number of authentication failures	10 or less	setMaximumFailedPasswordsForWipe()	

Lock screen	Inactivity to lockout	Time before lock screen is engaged	No required value	setMaximumTimeToLock()	To set an inactivity lockout go to Settings>Security & location> and tap on the gear icon next to Screen lock then tap on Automatically lock and select the appropriate value
	Banner	Banner message displayed on the lockscreen	Administrator or user defined text	setDeviceOwnerLockScreenInfo	To set a banner go to Settings>Security & location>Lock screen preferences>Lock screen message. Set a message and tap **Save**
	Remote Lock	Locks the device remotely	Function must be available	lockNow()	Tap the power button to turn off the screen which locks the device
	Show Password	Disallows the displaying of the password on the screen of lock-screen password	Disable	This is disabled by default	
	Notifications	Controls whether notifications are displayed on the lockscreen	Enable/Disable are available options	KEYGUARD_DISABLE_SECURE_NOTIFICATIONS() KEYGUARD_DISABLE_UNREDACTED_NOTIFICATIONS	
	Control Biometric Fingerprint	Control the use of Biometric Fingerprint authentication factor	Enable/Disable are available options	KEYGUARD_DISABLE_FINGERPRINT()	
Certificate Management	Import CA Certificates	Import CA Certificates into the Trust Anchor Database or the credential storage	No required value	installCaCert()	Tap on Settings>Security & location>Advanced>Encryption & credentials and select **Install from storage**
	Remove Certificates	Remove certificates from the Trust Anchor Database or the credential storage	No required value	uninstallCACert()	To clear all user installed certificates tap on Settings>Security & location>Advanced>Encryption & credentials and select Clear credentials To remove a specific user installed certificate tap on Settings>Security & location>Advanced>Enc

					ryption & credentials>Trusted credentials. Switch to the User tab, select the certificate you want to delete and tap on **Remove**
	Import Client Certificates	Import client certificates into Keychain	No required value	installKeyPair()	Tap on Settings>Security & location>Advanced>Encryption & credentials and select **Install from storage**
	Remove Client Certificates	Remove client certificates from Keychain	No required value	removeKeyPair()	To remove a specific user installed client certificate tap on Settings>Security & location>Advanced>Encryption & credentials>User credentials. Switch to the User tab, select the certificate you want to delete and tap on **Remove**
Radio Control	Control Wi-Fi	Control access to Wi-Fi	Enable/Disable are available options	DISALLOW_CONFIG_WIFI()	To disable Wi-Fi, tap on Settings>Network & internet and toggle Airplane mode to **On**
	Control GPS	Control access to GPS	Enable/Disable are available options	DISALLOW_SHARE_LOCATION() DISALLOW_CONFIG_LOCATION()	
	Control Cellular	Control access to Cellular	Enable/Disable are available options	DISALLOW_CONFIG_MOBILE_NETWORKS()	To disable Cellular tap on Settings>Network & internet>Mobile network and tap on your carrier and toggle to **Off**
	Control NFC	Control access to NFC	Enable/Disable are available options	DISALLOW_OUTGOING_BEAM()	To disable NFC tap on Settings>Connected devices>Connection preferences and toggle NFC to **Off**
	Control Bluetooth	Control access to Bluetooth	Enable/Disable are available options	DISALLOW_BLUETOOTH () DISALLOW_BLUETOOTH_SHARING() DISALLOW_CONFIG_BLUETOOTH()	
	Control Location Service	Control access to Location Service	Enable/Disable are available options	DISALLOW_SHARE_LOCATION() DISALLOW_CONFIG_LOCATION()	

Wi-Fi Settings	Specify Wi-Fi SSIDs	Specify SSID values for connecting to Wi-Fi. Can also create white and black lists for SSIDs.	No required value	WifiEnterpriseConfig()	
	Set WLAN CA Certificate	Select the CA Certificate for the Wi-Fi connection	No required value	WifiEnterpriseConfig()	
	Specify security type	Specify the connection security (WEP, WPA2, etc.)	No required value	WifiEnterpriseConfig()	
	Select authentication protocol	Specify the EAP-TLS connection values	No required value	WifiEnterpriseConfig()	
	Select client credentials	Specify the client credentials to access a specified WLAN	No required value	WifiEnterpriseConfig()	
	Control Always-on VPN	Control access to Always-on VPN	Enable/Disable are available options	setAlwaysOnVPNPackage()	
Hardware Control	Control Microphone	Control access to microphones	Enable/Disable are available options	DISALLOW_UNMUTE_MICROPHONE()	
	Control Camera	Control access to camera	Enable/Disable are available options	setCameraDisabled()	
	Control USB Mass Storage	Control access to mounting the device for storage over USB.	Enable/Disable are available options	DISALLOW_MOUNT_PHYSICAL_MEDIA()	
	Control USB Debugging	Control access to USB debugging.	Enable/Disable are available options	DISALLOW_DEBUGGING_FEATURES()	
	Control USB Tethered Connections	Control access to USB tethered connections.	Enable/Disable are available options	DISALLOW_CONFIG_TETHERING()	
	Control Bluetooth Tethered Connections	Control access to Bluetooth tethered connections.	Enable/Disable are available options	DISALLOW_CONFIG_TETHERING()	

	Control Hotspot Connections	Control access to Wi-Fi hotspot connections	Enable/Disable are available options	DISALLOW_CONFIG_TETHERING()	
	Automatic Time	Allows the device to get time from the Wi-Fi connection	Enable/Disable are available options	setAutoTimeRequired()	Tap on Settings>System>Date & time and toggle Automatic date & time to On
Application Control	Install Application	Installs specified application	No required value	PackageInstaller.Session()	
	Uninstall Application	Uninstalls specified application	No required value	uninstall()	To uninstall an application, tap on Settings>Applications & notifications>See all. Select the application and tap on Uninstall
	Application Whitelist	Specifies a list of applications that may be installed	No required value	This is done by the EMM/MDM when they setup an application catalog which leverages PackageInstaller.Session()	
	Application Blacklist	Specifies a list of applications that may not be installed	No required value	PackageInstaller.SessionInfo()	
	Application Repository	Specifies the location from which applications may be installed	No required value	DISALLOW_INSTALL_UNKNOWN_SOURCES()	
TOE Management	Enrollment	Enroll TOE in management	No required value		During device setup scan EMM/MDM provided QR code or enter EMM/MDM DPC identifier Refer to section 2.5.2 for more details
	Unenrollment	Unenroll TOE from management	No required value		To unenroll a factory reset must be performed on the device by going to Settings>System>Reset options and select **Erase all data (factory reset)**
	Allow Developer Mode	Controls Developer Mode access	Enable/Disable are available options	DISALLOW_DEBUGGING_FEATURES()	

	Sharing Data Between Enterprise and Personal Apps	Controls data sharing between enterprise and work apps	Enable/Disable	DISALLOW_CROSS_PROFILE_COPY_PASTE() addCrossProfileIntentFilter()	

3.5 Password Recommendations

When setting a password, you should select a password that:

- Does not use known information about yourself (such as, pet's names, your name, kid's names or any information available in the public domain).
- Is significantly different from previous passwords (adding a '1' or "!" to the end of the password is not sufficient).
- Does not contain a complete word. (Password!).
- Does not contain repeating or sequential numbers and/or letters.
- Follows the DoD password guidelines.

3.6 Honeywell Bug and Security Reporting Process

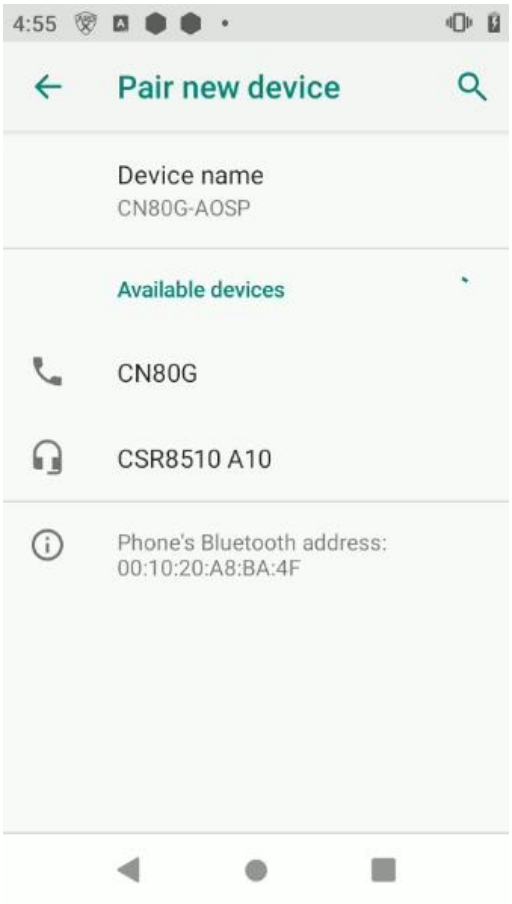
Honeywell Mobility edge devices
Honeywell provides a Product Security Incident response page where customers are able to report security findings, bugs, or any other potential security related issues. The web site is provided below and describes the methodology for submitting findings as well as the process which Honeywell follows to remediate them: https://www.honeywell.com/us/en/product-security

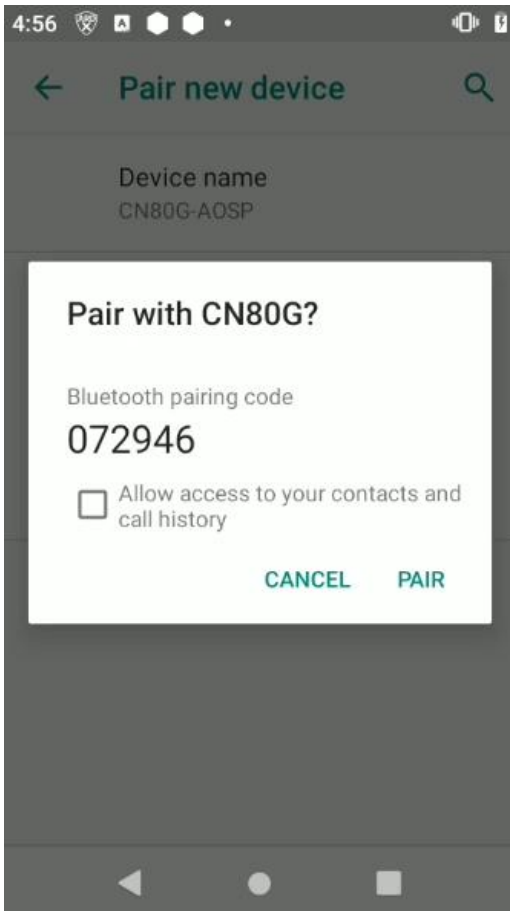
4. Bluetooth Configuration

Follow the below steps to pair and connect using Bluetooth

Pair

1. Open your device's Settings app .
2. Tap Connected devices > Connection preferences > Bluetooth. Make sure Bluetooth is turned on.
3. Tap Pair new device.
4. Tap the name of the Bluetooth device you want to pair with your phone or tablet.
5. Follow any on-screen steps.





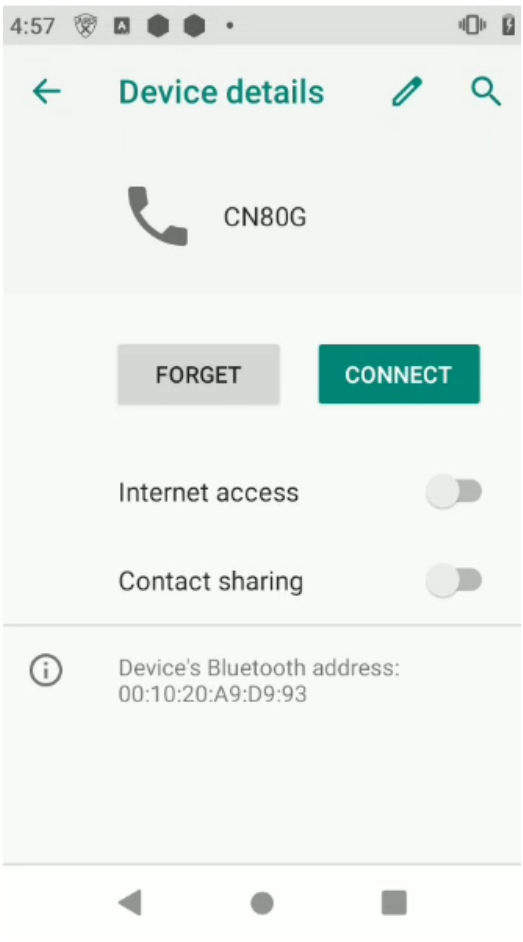
Connect

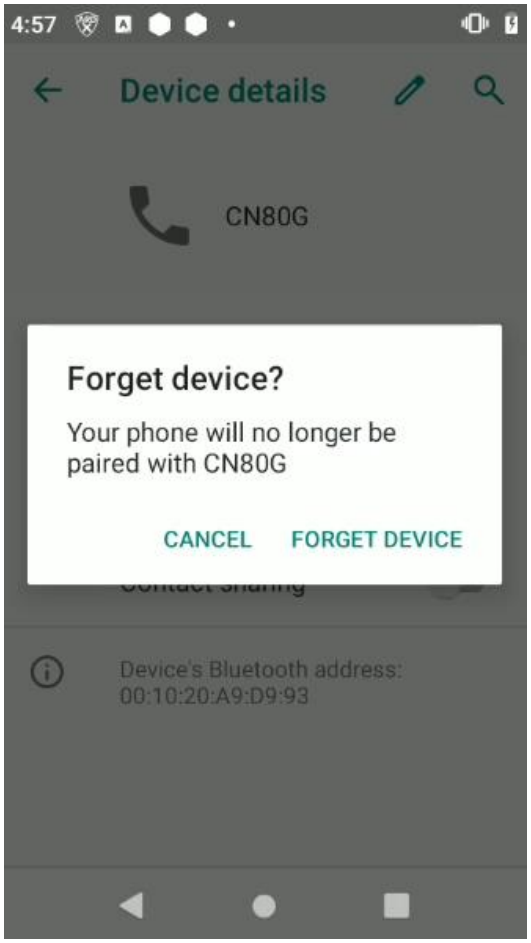
1. Open your device's Settings app .
2. Tap Connected devices > Connection preferences > Bluetooth.
3. Make sure Bluetooth is turned on.
4. In the list of paired devices, tap a paired but unconnected device.
5. When your Honeywell device and the Bluetooth device are connected, the device shows as "Connected" in the list.

Tip: If your Honeywell device is connected to something through Bluetooth, at the top of the screen, you'll see a Bluetooth icon .

Remove Previously Paired Device

1. Open your device or Settings app .
2. Tap Connected devices > Previously connected devices
3. Tap the gear icon to the right of the device you want to unpair
4. Tap on Forget and confirm in the popup window by tapping on Forget device





For additional support information around Bluetooth please take a look at this support link.

5. Wi-Fi Configuration

Android supports the WPA2-Enterprise (802.11i) protocol, which is specifically designed for enterprise networks and can be integrated into a broad range of Remote Authentication Dial-In User Service (RADIUS) authentication servers.

Administrators can silently provision enterprise Wi-Fi configurations on managed devices, including:

- SSID, via the EMM's DPC
- Password, via the EMM's DPC
- Identity, via the EMM's DPC
- Certificate for clients authorization, via the EMM's DPC
- CA certificate(s), via the EMM's DPC

Administrators can lock down Wi-Fi configurations on managed devices, to prevent users from creating new configurations or modifying configurations.

Administrators can lock down Wi-Fi configurations in either of the following configurations:

- Users cannot modify any Wi-Fi configurations provisioned by the EMM, but may add and modify their own user-configurable networks (for instance personal networks).

Users cannot add or modify any Wi-Fi network on the device, limiting Wi-Fi connectivity to just those networks provisioned by the EMM.

If a WiFi connection unintentionally terminates, the end user will need to reconnect to re-establish the session.

6. VPN Configuration

Android supports securely connecting to an enterprise network using VPN:

- **Always-on VPN**—The VPN can be configured so that apps don't have access to the network until a VPN connection is established, which prevents apps from sending data across other networks.
  - Always-on VPN supports VPN clients that implement VpnService. The system automatically starts that VPN after the device boots.
  - Device owners can direct all traffic to always be routed through a specified VPN.
  - Additionally, users can manually set Always-on VPN clients that implement VpnService methods using **Settings>More>VPN**. Always-on VPN can also be enabled manually from the settings menu.
- **Enterprise Always-on VPN** – The VPN can be configured so that any application inside of the work profile doesn't have access to the network until a VPN connection is established, which prevents apps from sending data across other networks.
  - Outside of isolating traffic originating from or destined to the work profile (and ignore traffic to and from the personal profile), this feature behaves identically to the base Always-on VPN feature.
  - Device owners can direct work apps inside of the Android Enterprise (Work) profile to always connect through a specified VPN.
  - Additionally, users can manually set Always-on VPN clients that implement VpnService methods using, from within the Android Enterprise Profile, **Settings>More>VPN**. Always-on VPN can also be enabled manually from the settings menu.

7. Work Profile Separation

Work profile mode is initiated when the DPC initiates a managed provisioning flow. The work profile is based on the Android multi-user concept, where the work profile functions as a separate Android user segregated from the primary profile. The work profile shares common UI real estate with the primary profile. Apps, notifications, and widgets from the work profile show up next to their counterparts from the primary profile and are always badged so users have an indication as to what type of app it is.

With the work profile, enterprise data does not intermix with personal application data. The work profile has its own apps, its own downloads folder, its own settings, and its own KeyChain. It is encrypted using its own encryption key, and it can have its own passcode to gate access.

The work profile is provisioned upon installation, and the user can only delete it by removing the entire work profile. Administrators can also remotely instruct the device policy client to remove the work profile, for instance, when a user leaves the organization, or a device is lost. Whether the user or an administrator removes the work profile, user data in the primary profile remains on the device.

A DPC running in profile owner mode can require users to specify a security challenge for apps running in the work profile. The system shows the security challenge when the user attempts to open any work apps. If the user successfully completes the security challenge, the system unlocks the work profile and decrypts it, if necessary.

Android also provides support for a separate work challenge to enhance security and control. The work challenge is a separate passcode that protects work apps and data. Admins managing the work profile can choose to set the password policies for the work challenge differently from the policies for other device passwords. Admins managing the work profile set the challenge policies using the usual DevicePolicyManager methods, such as setPasswordQuality() and setPasswordMinimumLength(). These admins can also configure the primary device lock, by using the DevicePolicyManager instance returned by the DevicePolicyManager.getParentProfileInstance() method.

As with the primary profile, the work challenge is verified within secure hardware, ensuring that it's difficult to brute-force. The passcode, mixed in with a secret from the secure hardware, is used to derive the disk encryption key for the work profile, which means that an attacker cannot derive the encryption key without either knowing the passcode or breaking the secure hardware.

Honeywell recommends the corporate owned personal enabled (COPE) STIG is followed and implemented in order for the device to properly be configured for a work-profile separation.

8. Secure Update Process

Local firmware updates (which includes, among others, baseband processor updates) are signed by a public key that is ultimately protected by the OEM Public Key, a hardware protected key whose SHA-256 hash resides inside the application processor. Should this verification fail, the software update fails, and the update is not installed. Additionally, Honeywell Mobility Edge devices also provide roll-back protection for local updates which prevent users from installing a previous version of software.

Honeywell Mobility Edge devices leverage "A/B system updates", also known as seamless updates. This approach ensures that a workable booting system image remains on the disk during the local update. This approach reduces the likelihood of an inactive device after an update, or an insecure firmware from which a device boots.

Administrators must copy the firmware update package to Honeywell Mobility Edge devices or push it through EMM console.

9. Audit Logging

An MDM agent acting as Device Owner can control the logging with DevicePolicyManager#setSecurityLoggingEnabled. When security logs are enabled, device owner apps receive periodic callbacks from DeviceAdminReceiver#onSecurityLogsAvailable, at which time a new batch of logs can be collected via DevicePolicyManager#retrieveSecurityLogs. SecurityEvent describes the type and format of security logs being collected.

The table below provides audit events:

Requirement	Auditable Events	Additional Audit Record Contents	Log Events & Examples
FAU_GEN.1	Start-up and shutdown of the audit functions		<Keyword> (<Date><Timestamp>): <message> Start-up: LOGGING_STARTED (Thu Sep 27 14:10:17 EDT 2018): Shutdown: All logs are stored in memory. When audit functions are disabled, all memory being used by the audit functions is released by the OS, and so this log cannot be seen.
	All administrative actions	**See Management Function Audits sheet**	
	Start-up and shutdown of the Rich OS		<Keyword> (<Date><Timestamp>): <message> Start-up: OS_STARTUP (Thu Sep 27 14:36:30 EDT 2018): orange enforcing Shutdown: All logs are stored in memory. This log is not capturable or persistent through boot, and thus isn't available to an MDM Administrator.
FCS_CKM_EXT.1	[*None*]	No additional information	
FCS_CKM_EXT.5	[*None*]	No additional information	
FCS_CKM.1	[*None*]	No additional information	
FCS_STG_EXT.1	Import or destruction of key	Identity of key. Role and identity of requestor	<Keyword> (<Date><Timestamp>): <message> KEY_IMPORTED (Tue Mar 19 17:25:52 EDT 2019): 1 USRPKEY_Test-Cert 1000
	[*No other events*]		<Keyword> (<Date><Timestamp>): <message> KEY_DESTROYED (Thu Sep 27 15:02:02 EDT 2018): 1 USRPKEY_android_pay_recent_unlock_key_2 10007
FCS_STG_EXT.3	Failure to verify integrity of stored key	Identity of key being verified	<Keyword> (<Date><Timestamp>): <message> KEY_INTEGRITY_VIOLATION (Fri Mar 22 20:38:19 EDT 2019):

			USRPKEY_f5c08a18548827f10f70006a7f0aa00b0902a75a 1000
FDP_DAR_EXT.1	[*None*]	No additional information	N/A
FDP_DAR_EXT.2	Failure to encrypt/decrypt data	No additional information	<Date> <Time> <ID> | <Keyword> <Message> 06-15 11:37:30.672 6748 6902 E CipherCompat: Failure to decrypt sensitive data: GeneralSecurityException: Keystore operation failed while trying to get RSA private key 06-15 11:39:20.142 6748 6902 E CipherCompat: Failure to encrypt sensitive data: GeneralSecurityException: Keystore operation failed while trying to get RSA public key
FDP_STG_EXT.1	Addition or removal of certificate from Trust Anchor Database	Subject name of certificate	<Keyword> (<Date><Timestamp>): <message> CERT_AUTHORITY_INSTALLED (Thu Sep 27 15:52:36 EDT 2018): 1 cn=certname rsa root ca,1.2.840.113549.1.9.1=#161a726f6f7463612d72736140676f7373616d65727365632e636f6d,O=ORG,l=catonsville,st=md,c=us
FIA_X509_EXT.1	Failure to validate X.509v3 certificate	Reason for failure of validation	<Date> <Time> <ID> | <Keyword> <Message> 02-21 19:12:35.488 1942 1973 W System.err: javax.net.ssl.SSLHandshakeException: java.security.cert.CertPathValidatorException: Trust anchor for certification path not found. 03-22 18:56:23.043 31552 31571 W System.err: java.security.cert.CertPathValidatorException: Certificate has been revoked, reason: UNSPECIFIED, revocation date: Tue Jan 15 12:30:07 EST 2019, authority: emailaddress=server-ocsp-subsubca-rsa@testsite.com, CN=server-ocsp-subsubca-rsa, O=ORG, L=Catonsville, ST=MD, C=US, extension OIDs: []
FMT_SMF_EXT.2	[None]	[None]	
FPT_NOT_EXT.1	[*None*]	[*No additional information*]	
FPT_TST_EXT.1	Initiation of self-test	[None]	<Keyword> (<Date><Timestamp>): <message> CRYPTO_SELF_TEST_COMPLETED (Thu Sep 27 14:10:17 EDT 2018): 1
	Failure of self-test		<Keyword> (<Date><Timestamp>): <message> CRYPTO_SELF_TEST_COMPLETED (Wed Mar 20 23:13:45 EDT 2018): 0
FPT_TST_EXT.2(1) (Selection is optional)	Start-up of TOE	No additional information	<Keyword> (<Date><Timestamp>): <message> OS_STARTUP (Thu Sep 27 14:36:30 EDT 2018): orange enforcing
	[None]	No additional information	
WLAN EP Audit Logs:			

FCS_TLSC_EXT.1/ WLAN	Failure to establish an EAP-TLS session	Reason for failure	<Date> <Time> <ID> | <Keyword> <Message> 03-20 18:25:46.017 10553 10553 W wpa_supplicant: TLS: Certificate verification failed, error 26 (unsupported certificate purpose) depth 0 for '/C=US/ST=MD/L=Catonsville/O=ORG/CN=tl9-16x.testsite.com/emailAddress=server-no-auth-eku-rsa@testsite.com'
	Establishment/termination of an EAP-TLS session	Non-TOE endpoint of connection	<Date> <Time> <ID> | <Keyword> <Message> Establishment: 03-20 18:18:38.619 9135 9135 I wpa_supplicant: wlan0: CTRL-EVENT-EAP-METHOD EAP vendor 0 method 13 (TLS) selected Termination: 03-20 18:28:56.358 10834 10834 I wpa_supplicant: wlan0: CTRL-EVENT-DISCONNECTED bssid=9c:4e:36:87:88:2c reason=1 locally_generated=1
FPT_TST_EXT.1/ WLAN (note: can be performed by TOE or TOE platform)	Execution of this set of TSF self-tests. [None]	[no additional information]	<Date> <Time> <ID> | <Keyword> <Message> Performed as part of MDFPP self-tests
FTA_WSE_EXT.1	All attempts to connect to access points		03-20 18:18:38.487 9135 9135 I wpa_supplicant: wlan0: Trying to associate with SSID 'nanoPC-EAP'
FTP_ITC_EXT.1/ WLAN	All attempts to establish a trusted channel	Identification of the non-TOE endpoint of the channel	Same as above

The table below provides sample management function audits:

REQUIREMENT	FUNCTION	Required Value	AUDIT LOG
FMT_SMF_EXT.1.1 Function 1	Configure password policy		
FMT_SMF_EXT.1.1 Function 1a	a. minimum password length	Greater than or equal to 8	<Keyword> (<Date><Timestamp>): <message> PASSWORD_COMPLEXITY_SET (Thu Mar 21 00:32:16 EDT 2019): com.afwsamples.testdpc 0 0 8 393216 1 0 1 0 0 1
FMT_SMF_EXT.1.1 Function 1b	b. minimum password complexity	No required value	<Keyword> (<Date><Timestamp>): <message> PASSWORD_COMPLEXITY_SET (Wed Oct 10 14:53:19 EDT 2018): com.afwsamples.testdpc 0 0 0 65536 1 0 1 0 0 1
FMT_SMF_EXT.1.1 Function 1c	c. maximum password lifetime		<Keyword> (<Date><Timestamp>): <message> PASSWORD_EXPIRATION_SET (Wed Oct 10 14:53:50 EDT 2018): com.afwsamples.testdpc 0 0 100000

FMT_SMF_EXT.1.1 Function 2	Configure session locking policy	10 minutes or less	
FMT_SMF_EXT.1.1 Function 2a	a. screen-lock enabled/disabled	Enabled	<Keyword> (<Date><Timestamp>): <message> PASSWORD_COMPLEXITY_SET (Wed Oct 10 14:53:19 EDT 2018): com.afwsamples.testdpc 0 0 0 65536 1 0 1 0 0 1
FMT_SMF_EXT.1.1 Function 2a	a. screen-lock enabled/disabled (after requiring a password above, admin can request the user set a password)	No required value	<Date> <Time> <ID> | <Keyword> <Message> 04-04 17:21:03.110 1173 5842 I ActivityManager: START u0 {act=android.app.action.SET_NEW_PASSWORD cmp=com.android.settings/.password.SetNewPasswordActivity} from uid 10160
FMT_SMF_EXT.1.1 Function 2a	a. screen-lock enabled/disabled (after requiring a password above, admin can forcibly set a password)		#<Intent ID>: <Action> <Additional information/Message> #64: act=android.app.action.ACTION_PASSWORD_CHANGED flg=0x10 (has extras)
FMT_SMF_EXT.1.1 Function 2b	b. screen lock timeout	10 minutes or less	<Keyword> (<Date><Timestamp>): <message> MAX_SCREEN_LOCK_TIMEOUT_SET (Wed Oct 10 14:42:18 EDT 2018): com.afwsamples.testdpc 0 0 10000
FMT_SMF_EXT.1.1 Function 2b	b. screen lock timeout (after setting a max time, the admin can prevent any user changes with this)		<Keyword> (<Date><Timestamp>): <message> USER_RESTRICTION_ADDED (Wed Oct 10 14:56:10 EDT 2018): com.afwsamples.testdpc 0 no_config_screen_timeout
FMT_SMF_EXT.1.1 Function 2c	c. number of authentication failures	10 or less	<Keyword> (<Date><Timestamp>): <message> MAX_PASSWORD_ATTEMPTS_SET (Wed Oct 10 14:42:26 EDT 2018): com.afwsamples.testdpc 0 0 20
FMT_SMF_EXT.1.1 Function 3	Enable/disable the VPN protection	Enable/ Disable	
FMT_SMF_EXT.1.1 Function 3a	a. across device		<Keyword> (<Date><Timestamp>): <message> USER_RESTRICTION_ADDED (Wed Oct 10 14:49:57 EDT 2018): com.afwsamples.testdpc 0 no_config_vpn
FMT_SMF_EXT.1.1 Function 4	enable/disable [assignment: list of all radios]		
FMT_SMF_EXT.1.1 Function 4a	Enable/disable [Bluetooth]	Enable/ Disable	<Keyword> (<Date><Timestamp>): <message> USER_RESTRICTION_ADDED (Wed Oct 10 14:43:06 EDT 2018): com.afwsamples.testdpc 0 no_bluetooth

FMT_SMF_EXT.1.1 Function 4b	Enable/disable [GPS]	Enable/ Disable	<Keyword> (<Date><Timestamp>): <message> USER_RESTRICTION_ADDED (Wed Oct 10 14:47:22 EDT 2018): com.afwsamples.testdpc 0 no_config_location USER_RESTRICTION_REMOVED (Wed Oct 10 16:33:02 EDT 2018): com.afwsamples.testdpc 0 no_config_location
FMT_SMF_EXT.1.1 Function 4c	Enable/disable [NFC]	Enable/ Disable	<Keyword> (<Date><Timestamp>): <message> USER_RESTRICTION_ADDED (Wed Oct 10 14:46:52 EDT 2018): com.afwsamples.testdpc 0 no_outgoing_beam
FMT_SMF_EXT.1.1 Function 4d	Enable/disable [Wi-Fi]	Enable/ Disable	User feature only
FMT_SMF_EXT.1.1 Function 4e	Enable/disable [Cellular/Mobile data]	Enable/ Disable	User feature only
FMT_SMF_EXT.1.1 Function 5a	Enable/disable [microphone] a. across device	Enable/ Disable	<Keyword> (<Date><Timestamp>): <message> USER_RESTRICTION_ADDED (Wed Oct 10 14:55:44 EDT 2018): com.afwsamples.testdpc 0 no_unmute_microphone
FMT_SMF_EXT.1.1 Function 5b	Enable/disable [camera, microphone] c. On a per-app basis	Enable/ Disable	User feature only
FMT_SMF_EXT.1.1 Function 6	Transition to the locked state		<Keyword> (<Date><Timestamp>): <message> REMOTE_LOCK (Thu Oct 11 11:34:16 EDT 2018): com.afwsamples.testdpc 0 0
FMT_SMF_EXT.1.1 Function 7	Full wipe of protected data		The act of wiping protected data resets the phone, and no audit record available. Wiping of phone available to the administrator via an MDM API and to the user using factory reset.
FMT_SMF_EXT.1.1 Function 8a	Configure application installation policy a. restricting the sources of applications	Disable	<Keyword> (<Date><Timestamp>): <message> USER_RESTRICTION_REMOVED (Wed Oct 10 14:47:00 EDT 2018): com.afwsamples.testdpc 0 no_install_unknown_sources
FMT_SMF_EXT.1.1 Function 8a	Configure application installation policy a. restricting the sources of applications	Enable	<Keyword> (<Date><Timestamp>): <message> USER_RESTRICTION_ADDED (Wed Oct 10 14:47:00 EDT 2018): com.afwsamples.testdpc 0 no_install_unknown_sources
FMT_SMF_EXT.1.1 Function 8c	Configure application installation policy c. denying	Enable	<Keyword> (<Date><Timestamp>): <message> USER_RESTRICTION_ADDED (Wed Oct 10 14:47:03 EDT 2018): com.afwsamples.testdpc 0 no_install_apps

	installation of applications		
FMT_SMF_EXT.1.1 Function 9	Import keys/secrets to secure key store		<Keyword> (<Date><Timestamp>): <message> KEY_IMPORTED (Fri Mar 08 11:37:39 EST 2019): 1 USRPKEY_client-rsa 1000
FMT_SMF_EXT.1.1 Function 10	Destroy imported keys/secrets in secure key store		<Keyword> (<Date><Timestamp>): <message> KEY_DESTROYED (Fri Mar 08 11:37:43 EST 2019): 1 USRPKEY_client-rsa 1000
FMT_SMF_EXT.1.1 Function 11	Import X.509v3 certificates into the Trust Anchor Database		<Keyword> (<Date><Timestamp>): <message> CERT_AUTHORITY_INSTALLED (Wed Oct 10 14:45:25 EDT 2018): 1 cn=certname rsa root ca,1.2.840.113549.1.9.1=#161a726f6f7463612d72736140676f7373616d65727365632e636f6d,O=ORG,l=catonsville,st=md,c=us
FMT_SMF_EXT.1.1 Function 12	Remove imported X.509v3 certificates and [no other certificates] from the Trust Anchor Database		<Keyword> (<Date><Timestamp>): <message> CERT_AUTHORITY_REMOVED (Wed Oct 10 14:45:31 EDT 2018): 1 cn=certname rsa root ca,1.2.840.113549.1.9.1=#161a726f6f7463612d72736140676f7373616d65727365632e636f6d,O=ORG,l=catonsville,st=md,c=us
FMT_SMF_EXT.1.1 Function 13	Enroll TOE in management		Logs do not exist until after the MDM agent has been installed and enabled security logging.
FMT_SMF_EXT.1.1 Function 14	Remove apps		#<Intent ID>: <Action> <Additional information/Message> #126: act=android.intent.action.PACKAGE_REMOVED dat=package:org.fdroid.fdroid flg=0x4000010 (has extras) +1ms dispatch +3ms finish enq=2019-03-18 11:31:55 disp=2019-03-18 11:31:55 fin=2019-03-18 11:31:55 extras: Bundle[{android.intent.extra.REMOVED_FOR_ALL_USERS=false, android.intent.extra.DONT_KILL_APP=false, android.intent.extra.UID=10172, android.intent.extra.DATA_REMOVED=false, android.intent.extra.user_handle=0}]
FMT_SMF_EXT.1.1 Function 15	Update system software		06-14 12:56:03.444 18996 20854 I SystemUpdate: [Execution,InstallationIntentOperation] Received intent: Intent { act=com.google.android.gms.update.INSTALL_UPDATE cat=[targeted_intent_op_prefix:.update.execution.InstallationIntentOperation] cmp=com.google.android.gms/.chimera.GmsIntentOperationService }.

FMT_SMF_EXT.1.1 Function 16	Install apps		#<Intent ID>: <Action> <Additional information/Message> #110: act=android.intent.action.PACKAGE_ADDED dat=package:org.fdroid.fdroid flg=0x4000010 (has extras) +2ms dispatch +2ms finish enq=2019-03-18 11:23:59 disp=2019-03-18 11:23:59 fin=2019-03-18 11:23:59 extras: Bundle[{android.intent.extra.UID=10172, android.intent.extra.user_handle=0}]
FMT_SMF_EXT.1.1 Function 17	Remove Enterprise apps		#<Intent ID>: <Action> <Additional information/Message> #259: act=android.intent.action.PACKAGE_REMOVED dat=package:org.fdroid.fdroid flg=0x4000010 pkg=com.afwsamples.testdpc (has extras) +3ms dispatch 0 finish enq=2019-03-18 11:23:59 disp=2019-03-18 11:23:59 fin=2019-03-18 11:23:59 extras: Bundle[{android.intent.extra.REMOVED_FOR_ALL_USERS=false, android.intent.extra.DONT_KILL_APP=false, android.intent.extra.UID=1210172, android.intent.extra.DATA_REMOVED=false, android.intent.extra.REPLACING=true, android.intent.extra.user_handle=12}]
FMT_SMF_EXT.1.1 Function 18	Configure the Bluetooth trusted channel		
FMT_SMF_EXT.1.1 Function 18a	a. disable/enable the Discoverable mode (for BR/EDR)		User feature only
FMT_SMF_EXT.1.1 Function 18b	b. change the Bluetooth device name		User feature only
FMT_SMF_EXT.1.1 Function 19	Enable/disable notifications in locked state f. all notifications	Enable/ Disable	<Keyword> (<Date><Timestamp>): <message> KEYGUARD_DISABLED_FEATURES_SET (FRI MAR 08 14:01:02 EST 2019): com.afwsamples.testdpc 0 0 <0/4/8/12> 0 = no restriction 4 = disable unredated notifications ("Show all notification content" setting selection greyed out) 8 = disable secure notifications ("Hide sensitive content" & "Show all notification content" greyed out) 12 = both 4 and 8 (but 8 supersedes 4 anyway)
FMT_SMF_EXT.1.1 Function 20	Enable DAR protection		N/A - DAR cannot be disabled
FMT_SMF_EXT.1.1 Function 22	Enable/disable location services	Enable/ Disable	

FMT_SMF_EXT.1.1 Function 22a	a. across device		<Keyword> (<Date><Timestamp>): <message> USER_RESTRICTION_ADDED (Wed Oct 10 14:47:13 EDT 2018): com.afwsamples.testdpc 0 no_share_location
FMT_SMF_EXT.1.1 Function 23	Enable/disable the use of [Biometric Authentication Factor]	Enable	<Keyword> (<Date><Timestamp>): <message> KEYGUARD_DISABLED_FEATURES_SET (Wed Oct 10 14:57:53 EDT 2018): com.afwsamples.testdpc 0 0 420
FMT_SMF_EXT.1.1 Function 25	Enable/disable [Wi-Fi tethering, USB tethering, and Bluetooth tethering]	Disable	<Keyword> (<Date><Timestamp>): <message> USER_RESTRICTION_ADDED (Wed Oct 10 14:58:21 EDT 2018): com.afwsamples.testdpc 0 no_config_tethering USER_RESTRICTION_REMOVED (Wed Oct 10 16:37:47 EDT 2018): com.afwsamples.testdpc 0 no_config_tethering
FMT_SMF_EXT.1.1 Function 26	Enable/disable developer modes	Disable	<Keyword> (<Date><Timestamp>): <message> USER_RESTRICTION_ADDED (Thu Oct 11 11:48:28 EDT 2018): com.afwsamples.testdpc 0 no_debugging_features USER_RESTRICTION_REMOVED (Thu Oct 11 11:48:30 EDT 2018): com.afwsamples.testdpc 0 no_debugging_features
FMT_SMF_EXT.1.1 Function 28	Wipe Enterprise Data		#<Intent ID>: <Action> <Additional information/Message> #30: act=android.intent.action.MANAGED_PROFILE_REMOVED flg=0x50000010 (has extras)
FMT_SMF_EXT.1.1 Function 41	Enable/disable [Hotspot functionality, USB tethering]	Enable/ Disable	<Keyword> (<Date><Timestamp>): <message> USER_RESTRICTION_ADDED (Wed Oct 10 14:58:21 EDT 2018): com.afwsamples.testdpc 0 no_config_tethering
FMT_SMF_EXT.1.1 Function 44	Unenroll TOE from management		#<Intent ID>: <Action> <Additional information/Message> #14: act=android.app.action.DEVICE_ADMIN_DISABLED flg=0x10
FMT_SMF_EXT.1.1 Function 45	Enable/disable the Always On VPN protection	Enable	<Keyword> (<Date><Timestamp>): <message> SettingsProvider: Notifying for 0: content://settings/secure/always_on_vpn_app 03-28 10:37:05.091 1400 1400 VSettingsProvider: Notifying for 0: content://settings/secure/always_on_vpn_lockdown
FMT_SMF_EXT.1.1 Function 48	Configure Security Policy for each wireless network:		<Date> <Time> <ID> | <Keyword> <Message> 05-28 18:57:54.859 1220 4568 I addOrUpdateNetwork: uid = 10161 SSID "NetworkSSID" nid=-1 05-28 18:58:31.667 1220 1539 I addOrUpdateNetwork: uid = 10161 SSID NetworkEAPTLS nid=-1

FMT_SMF_EXT.1.1 Function 48a	a. [selection: specify the CA(s) from which the TSF will accept WLAN authentication server certificate(s), specify the FQDN(s) of acceptable WLAN authentication server certificate(s)]		See Function 48
FMT_SMF_EXT.1.1 Function 48b	b. security type		See Function 48
FMT_SMF_EXT.1.1 Function 48c	c. authentication protocol		See Function 48
FMT_SMF_EXT.1.1 Function 48d	d. client credentials		See Function 48
FMT_SMF_EXT.1.1 Function 49	specify wireless networks (SSIDs) to which the TSF may connect		See Function 48

10. FDP_DAR_EXT.2 - Sensitive Data Protection Overview

Using the NIAPSEC library, sensitive data protection is enabled by default by using the Strong configuration.

To request access to the NIAPSEC library, please reach out to Honeywell via:
https://www.honeywellaidc.com/resources/support

The library provides APIs via SecureContextCompat to write files when the device is either locked or unlocked. Reading an encrypted file is only possible when the device is unlocked and authenticated.

Saving sensitive data files requires a key to be generated in advance. Please see the Key generation section for more information.

Supported Algorithms via SecureConfig.getStrongConfig()

File Encryption Key: AES256 - AES/GCM/NoPadding

Key Encryption Key: RSA4096 - RSA/ECB/OAEPWithSHA-256AndMGF1Padding

Writing Encrypted (Sensitive) Files:

SecureContextCompat opens a FileOutputStream for writing and uses SecureCipher (below) to encrypt the data.

The Key Encryption Key, which is stored in the Android KeyStore, encrypts the File Encryption Key which is encoded with the file data.

Reading Encrypted (Sensitive) Files:

SecureContextCompat opens a FileInputStream for reading and uses SecureCipher (below) to decrypt the data.

The Key Encryption Key, which is stored in the AndroidKeystore, decrypts the File Encryption Key which is encoded with the file data.

The File encryption key material is automatically destroyed and removed from memory after each operation. Please see EphemeralSecretKey for more information.

10.1 SecureContextCompat

Included in the NIAPSEC library.
Encrypt and decrypt files that require sensitive data protection.

Supported Algorithms:
AES256 - AES/GCM/NoPadding
RSA4096 - RSA/ECB/OAEPWithSHA-256AndMGF1Padding

Public Constructor	
SecureContextCompat	**new SecureContextCompat**(Context, BiometricSupport) *See BiometricSupport* Constructor to create an instance of the SecureContextCompat with Biometric support.
Public Methods	
FileOutputStream	**openEncryptedFileOutput** (String name, **int** mode, String keyPairAlias) Gets an encrypted file output stream using the a*symmetric/ephemeral* algorithms specified by the default configuration, using NIAP standards. -name - The file name -mode - The file mode, usually Context.MODE_PRIVATE -keyPairAlias - Encrypt data with the AndroidKeyStore key referenced - Key Encryption Key
void	**openEncryptedFileInput** (String name, Executor executor, EncryptedFileInputStreamListener listener) Gets an encrypted file input stream using the a*symmetric/ephemeral* algorithms specified by the default configuration, using NIAP standards. -name - The file name -Executor - to handle the threading for BiometricPrompt. Usually Executors.newSingleThreadExecutor() -Listener for the resulting FileInputStream.

Code Examples:

```
SecureContextCompat secureContext = new SecureContextCompat(getApplicationContext(),
SecureConfig.getStrongConfig(biometricSupport));

// Open a sensitive file for writing
FileOutputStream outputStream = secureContext.openEncryptedFileOutput(FILE_NAME,
Context.MODE_PRIVATE, KEY_PAIR_ALIAS);
// Write data to the file, where DATA is a String of sensitive information.
outputStream.write(DATA.getBytes(StandardCharsets.UTF_8));
outputStream.flush();
outputStream.close();

// Read a sensitive data file
 secureContext.openEncryptedFileInput(FILE_NAME, Executors.newSingleThreadExecutor(),
inputStream -> {
   byte[] clearText = new byte[inputStream.available()];
   inputStream.read(encodedData);
   inputStream.close();
  // do something with the decrypted data
});
```

Built using the JCE libraries, for more information please see the following resources:

AndroidKeyStore - https://developer.android.com/training/articles/keystore

BiometricPrompt - https://developer.android.com/reference/android/hardware/biometrics/BiometricPrompt

11. API Specification

This section provides a list of the evaluated cryptographic APIs that developers can use when writing their mobile applications.

1. Cryptographic APIs
    - This section lists all the APIs for the algorithms and random number generation
2. Key Management
    - Need APIs for importing, using, and destroying keys
3. Certificate Validation, TLS, HTTPS
    - API used by applications for configuring the reference identifier
    - APIs for validation checks (should match the test program provided)
    - TLS, HTTPS, Bluetooth BR/EDR (any other protocol available to applications)

11.1 Cryptographic APIs

Code samples to do encryption and decryption, including random number generation.

Code examples:

```
// Data to encrypt
byte[] clearText = "Secret Data".getBytes(StandardCharsets.UTF_8);

// Create a Biometric Support object to handle key authentication
BiometricSupport biometricSupport = new BiometricSupportImpl(activity,
getApplicationContext()) {
        …
};

SecureCipher secureCipher = SecureCipher.getDefault(biometricSupport);
secureCipher.encryptSensitiveData("niapKey", clearText, new
SecureCipher.SecureSymmetricEncryptionCallback() {
  @Override
  public void encryptionComplete(byte[] cipherText, byte[] iv) {
    // Do something with the encrypted data
  }
});

// to decrypt
secureCipher.decryptSensitiveData("niapKey", cipherText, iv, new
SecureCipher.SecureDecryptionCallback() {
  @Override
  public void decryptionComplete(byte[] clearText) {
    // do something with the encrypted data
  }
});
```

// Generate ephemeral key (random number generation)

```
int keySize = 256;
SecureRandom secureRandom = SecureRandom.getInstanceStrong();
byte[] key = new byte[keySize / 8];
secureRandom.nextBytes(key);

// Encrypt / decrypt data with the ephemeral key
EphemeralSecretKey ephemeralSecretKey = new EphemeralSecretKey(key,
SecureConfig.getStrongConfig());
Pair<byte[], byte[]> ephemeralCipherText =
secureCipher.encryptEphemeralData(ephemeralSecretKey, clearText);
byte[] ephemeralClearText = secureCipher.decryptEphemeralData(ephemeralSecretKey,
ephemeralCipherText.first, ephemeralCipherText.second);
```

11.1.1 SecureCipher

Included in the NIAPSEC library.
Handles low-level cryptographic operations including encryption and decryption. For sensitive data protection this library is not used directly by developers.

Supported Algorithms:
AES256 - AES/GCM/NoPadding
RSA4096 - RSA/ECB/OAEPWithSHA-256AndMGF1Padding

Public Static Accessors	
SecureCipher	**SecureCipher.getDefault**(BiometricSupport) *See BiometricSupport* API to get an instance of the SecureCipher with Biometric support.
Public Methods	
void	**encryptSensitiveData** (String keyAlias, **byte**[] clearData, SecureSymmetricEncryptionCallback callback) Encrypt sensitive data using the *symmetric* algorithm specified by the default configuration, using NIAP standards. *See SecureConfig.getStrongConfig() - Default is AES256 GCM.* -keyAlias - Encrypt data with the AndroidKeyStore key referenced -clearData - the data to be encrypted -callback, the callback to return the cipherText after encryption is complete.
void	**encryptSensitiveDataAsymmetric** (String keyAlias, **byte**[] clearData, SecureAsymmetricEncryptionCallback callback) Encrypt sensitive data using the *asymmetric* algorithm specified by the default configuration, using NIAP standards. *See SecureConfig.getStrongConfig() - Default is RSA4096 with OAEP.* -keyAlias - Encrypt data with the AndroidKeyStore key referenced -clearData - the data to be encrypted -callback, the callback to return the cipherText after encryption is complete.

Pair<**byte**[], **byte**[]>	**encryptEphemeralData** (EphemeralSecretKey ephemeralSecretKey, byte[] clearData) Encrypt data with an Ephemeral AES 256 GCM key, used for encrypting file data for SDP. -The Ephemeral key to use -clearData, the data to be encrypted Returns a Pair of the cipherText, and IV byte arrays respectively.
void	**decryptSensitiveData** (String keyAlias, **byte**[] encryptedData, **byte**[] initializationVector, SecureDecryptionCallback callback) Decrypt sensitive data using the *symmetric* algorithm specified by the default configuration, using NIAP standards. *See SecureConfig.getStrongConfig() - Default is AES256 GCM.* -keyAlias - Encrypt data with the AndroidKeyStore key referenced -encryptedData - the data to be decrypted -initializationVector - the IV used for encryption -callback, the callback to return the clearText after decryption is complete.
void	**decryptSensitiveData** (String keyAlias, **byte**[] encryptedData, SecureDecryptionCallback callback) Decrypt sensitive data using the a*symmetric* algorithm specified by the default configuration, using NIAP standards. *See SecureConfig.getStrongConfig() - Default is RSA4096 with OAEP.* -keyAlias - Encrypt data with the AndroidKeyStore key referenced -encryptedData - the data to be decrypted -callback, the callback to return the clearText after decryption is complete.
byte[]	**decryptEphemeralData** (EphemeralSecretKey ephemeralSecretKey, **byte**[] encryptedData, **byte**[] initializationVector) Decrypt data with an Ephemeral AES 256 GCM key, used for encrypting file data for SDP. -The Ephemeral key to use -encryptedData - the data to be decrypted -initializationVector - the IV used for encryption Returns a byte array of the clear text.

Built using the JCE libraries for more information please see the following resources:

AndroidKeyStore - https://developer.android.com/training/articles/keystore
Cipher - https://developer.android.com/reference/javax/crypto/Cipher
SecretKey - https://developer.android.com/reference/javax/crypto/SecretKey
SecureRandom - https://developer.android.com/reference/java/security/SecureRandom

11.1.2 FCS_CKM.2(1) - RSA

Assume that Alice knows a private key and Bob knows Alice's public key. Bob sent a key encrypted by the public key. This example shows how Alice gets a plain key sent by Bob. Alice needs her own private key to decrypt an encrypted key.

```
KeyPairGenerator keyGen = KeyPairGenerator.getInstance("RSA", "AndroidOpenSSL");
keyGen.initialize(keySize);
KeyPair keyPair = keyGen.generateKeyPair();
RSAPublicKey pub = (RSAPublicKey) keyPair.getPublic();
RSAPrivateCrtKey priv = (RSAPrivateCrtKey) keyPair.getPrivate();

// Encrypt
Cipher cipher = Cipher.getInstance("RSA/ECB/OAEPWithSHA-256AndMGF1Padding");
cipher.init(Cipher.ENCRYPT_MODE, publicKey, new OAEPParameterSpec("SHA-256",
      "MGF1", new MGF1ParameterSpec("SHA-1"), PSource.PSpecified.DEFAULT));
byte[] cipherText = cipher.doFinal(data.getBytes(StandardCharsets.UTF_8));

// Decrypt
Cipher cipher = Cipher.getInstance("RSA/ECB/OAEPWithSHA-256AndMGF1Padding");
cipher.init(Cipher.DECRYPT_MODE, privateKey, new OAEPParameterSpec("SHA-256",
      "MGF1", new MGF1ParameterSpec("SHA-1"), PSource.PSpecified.DEFAULT));
Byte[] plainText = cipher.doFinal(cipherText);
```

Algorithms::
RSA/ECB/OAEPWithSHA-256AndMGF1Padding

Reference:
Cipher - https://developer.android.com/reference/javax/crypto/Cipher

11.1.3 FCS_CKM.2(1) – ECDSA (Signature)

Assume that Alice knows a private key and Bob's public key. Bob knows his private key and Alice's public key. Then Alice and Bob can sign/verify the contents of a message.

```
KeyPairGenerator keyGen = KeyPairGenerator.getInstance("EC", "AndroidOpenSSL");
ECGenParameterSpec ecParams = new ECGenParameterSpec(spec);
keyGen.initialize(ecParams);
KeyPair keyPair = keyGen.generateKeyPair();
ECPublicKey pubKey = (ECPublicKey) keyPair.getPublic();
ECPrivateKey privKey = (ECPrivateKey) keyPair.getPrivate();

// Sign
Signature signature = Signature.getInstance(algorithm);
signature.initSign(privateKey);
signature.update(data.getBytes(StandardCharsets.UTF_8));
byte[] signature =  signature.sign();
```

```
// Verify
Signature signature = Signature.getInstance(algorithm);
signature.initVerify(publicKey);
signature.update(data.getBytes(StandardCharsets.UTF_8));
boolean verified =  signature.verify(sig);
```

Algorithms:
"SHA256withECDSA", "secp256r1"
"SHA384withECDSA", "secp384r1"

Reference:
Signature - https://developer.android.com/reference/java/security/Signature

11.1.4 FCS_CKM.1 – ECDSA Key Establishment (Signature)

Assume that Alice knows a private key and Bob's public key. Bob knows his private key and Alice's public key.

```
KeyPairGenerator keyGen = KeyPairGenerator.getInstance("EC", "AndroidOpenSSL");
ECGenParameterSpec ecParams = new ECGenParameterSpec(spec);
keyGen.initialize(ecParams);
KeyPair keyPair = keyGen.generateKeyPair();
ECPublicKey pubKey = (ECPublicKey) keyPair.getPublic();
ECPrivateKey privKey = (ECPrivateKey) keyPair.getPrivate();
```

Algorithms:
"SHA256withECDSA", "secp256r1"
"SHA384withECDSA", "secp384r1"

Reference:
Signature - https://developer.android.com/reference/java/security/Signature

11.1.5 FCS_COP.1(1) - AES

Cipher class encrypts or decrypts a plain text.

```
KeyGenerator keyGenerator = KeyGenerator.getInstance("AES", "AndroidOpenSSL");
keyGenerator.init(keySize);
SecretKey key = keyGenerator.generateKey();

// Encrypt
Cipher cipher = Cipher.getInstance(transformation);
cipher.init(Cipher.ENCRYPT_MODE, secretKey);
byte[] iv = cipher.getIV();
byte[] clearData = data.getBytes(UTF_8);
byte[] cipherText = cipher.doFinal(clearData);
Pair<byte[], byte[]> result =  Pair<>(cipherText, iv);
```

```
// Decrypt
Cipher cipher = Cipher.getInstance(transformation);
cipher.init(Cipher.DECRYPT_MODE, secretKey, spec);
String plainText =  new String(cipher.doFinal(cipherText), UTF_8);
```

Algorithms:
AES/CBC/NoPadding
AES/GCM/NoPadding

Reference:
Cipher - https://developer.android.com/reference/javax/crypto/Cipher
FCS_COP.1(2) - SHA
You can use MessageDigest class to calculate the hash of plaintext.

```
MessageDigest messageDigest = MessageDigest.getInstance(algorithm);
messageDigest.update(data.getBytes(StandardCharsets.UTF_8));
byte[] digest = messageDigest.digest();
```

Algorithms:
SHA-1
SHA-256
SHA-384
SHA-512

Reference:
MessageDigest - https://developer.android.com/reference/java/security/MessageDigest

11.1.6 FCS_COP.1(3) – RSA (Signature Algorithms)

KeyFactory class generates RSA private key and public key. Signature class signs a plaintext with private key generated above and verifies it with public key

```
KeyPairGenerator keyGen = KeyPairGenerator.getInstance("RSA", "AndroidOpenSSL");
keyGen.initialize(keySize);
KeyPair keyPair = keyGen.generateKeyPair();
RSAPublicKey pub = (RSAPublicKey) keyPair.getPublic();
RSAPrivateCrtKey priv = (RSAPrivateCrtKey) keyPair.getPrivate();

// Sign
Signature signature = Signature.getInstance(algorithm);
signature.initSign(privateKey);
signature.update(data.getBytes(StandardCharsets.UTF_8));
byte[] sig =  signature.sign();

// Verify
Signature signature = Signature.getInstance(algorithm);
signature.initVerify(publicKey);
signature.update(data.getBytes(StandardCharsets.UTF_8));
```

```
boolean verified =  signature.verify(sig);
```

Algorithms:
SHA256withRSA
SHA384withRSA

Reference:
Signature - https://developer.android.com/reference/java/security/Signature

11.1.7 FCS_CKM.1 – RSA Key Establishment (Signature Algorithms)

KeyFactory class generates RSA private key and public key.

```
KeyPairGenerator keyGen = KeyPairGenerator.getInstance("RSA", "AndroidOpenSSL");
keyGen.initialize(keySize);
KeyPair keyPair = keyGen.generateKeyPair();
RSAPublicKey pub = (RSAPublicKey) keyPair.getPublic();
RSAPrivateCrtKey priv = (RSAPrivateCrtKey) keyPair.getPrivate();
```

Algorithms:
SHA256withRSA
SHA384withRSA

Reference:
Signature - https://developer.android.com/reference/java/security/Signature

11.1.8 FCS_COP.1(4) - HMAC

Mac class calculates the hash of plaintext with key.

```
KeyGenerator keyGenerator = KeyGenerator.getInstance(
      algorithm, "AndroidOpenSSL");
keyGenerator.init(keySize);
SecretKey key = keyGenerator.generateKey();

// Mac
Mac mac = Mac.getInstance(algorithm);
mac.init(secretKey);
byte[] mac =  mac.doFinal(data.getBytes(StandardCharsets.UTF_8));
```

Algorithms:
HmacSHA1
HmacSHA256
HmacSHA384
HmacSHA512

Reference:
Mac - https://developer.android.com/reference/javax/crypto/Mac

11.2 Key Management

Code samples to do key management.

Code examples:

```
SecureKeyGenerator keyGenerator = SecureKeyGenerator.getInstance();
// Generate Keypair
keyGenerator.generateAsymmetricKeyPair(KEY_PAIR_ALIAS);
// Generate Symmetric Key
keyGenerator.generateKey(KEY_ALIAS);

// Generate ephemeral key (random number generation)
keyGenerator.generateEphemeralDataKey();

// To delete a key stored in the Android Keystore
KeyStore keyStore = KeyStore.getInstance("AndroidKeyStore");
keyStore.load(null);
keyStore.deleteEntry("KEY_TO_REMOVE");
```

11.2.1 SecureKeyGenerator

Included in the NIAPSEC library.
Handles low-level key generation operations using the AndroidKeyStore. For sensitive data protection this library is not used directly by developers.

Supported Algorithms:
AES256 - AES/GCM/NoPadding
RSA4096 - RSA/ECB/OAEPWithSHA-256AndMGF1Padding

Public Static Accessors	
SecureKeyGenerator	**SecureCipher.getDefault**() API to get an instance of the SecureCipher with NIAP settings.
Public Methods	
boolean	generateKey(String keyAlias) Generate an AES key with NIAP settings that is stored and protected in the AndroidKeyStore. *See SecureConfig.getStrongConfig() - Default is AES256 GCM.* -keyAlias - name for the key
boolean	generateKeyAsymmetricKeyPair(String keyAlias) Generate an RSA key pair with NIAP settings that is stored and protected in the AndroidKeyStore. *See SecureConfig.getStrongConfig() - Default is RSA4096 OAEP.* -keyAlias - name for the key pair

EphemeralSecretKey	**generateEphemeralDataKey()** Generate an AES key with NIAP settings. This key is not stored in the AndroidKeyStore Uses SecureRandom.getInstanceStrong() to generate a random key. *See SecureConfig.getStrongConfig() - Default is AES256 GCM.*

Built using the JCE libraries, for more information please see the following resources:
AndroidKeyStore - https://developer.android.com/training/articles/keystore
KeyPairGenerator- https://developer.android.com/reference/java/security/KeyPairGenerator
SecretKey - https://developer.android.com/reference/javax/crypto/SecretKey
SecureRandom - https://developer.android.com/reference/java/security/SecureRandom
KeyGenParameterSpec - https://developer.android.com/reference/android/security/keystore/KeyGenParameterSpec

11.3 FCS_TLSC_EXT.1 - Certificate Validation, TLS, HTTPS

Included in the NIAPSEC library.
SecureURL automatically configures TLS and can perform certificate and host validation checking. At construction, SecureURL requires a reference identifier.

Code examples:

```
SecureURL url = new SecureURL(referenceIdentifier, "google_cert");
HttpsURLConnection conn = (HttpsURLConnection) url.openConnection();
conn.setRequestMethod("GET");
conn.setDoInput(true);
conn.connect();

// Manual check
SecureURL url = new SecureURL(referenceIdentifier, "google_cert");
boolean valid = url.isValid(urlConnection);
```

Public Constructors	
SecureURL	**new SecureURL(String referenceIdentifier, String clientCert)** API to create an instance of the SecureURL with NIAP settings. clientCert is optional.
Public Methods	
HttpsURLConnection	openConnection Opens an HttpsUrlConnection using TLS by default and handles OCSP validation checks and does a hostname verification check on initiation of the connection.
boolean	isValid(String hostname, SSLSocket socket) A manual OCSP certificate and hostname check. Based on a hostname and underlying SSLSocket.

boolean	isValid(HttpsURLConnection conn) A manual OCSP certificate and hostname check. Based on an existing HttpsUrlConnection.
boolean	isValid(Certificate cert) A manual OCSP certificate check.
boolean	isValid(List<Certificate> certs) A manual OCSP certificates check.

Built using the networking libraries for more information please see the following resources:
HttpsUrlConnection - https://developer.android.com/reference/javax/net/ssl/HttpsURLConnection
PKIXRevocationChecker - https://developer.android.com/reference/java/security/cert/PKIXRevocationChecker
SSLSocket - https://developer.android.com/reference/javax/net/ssl/SSLSocket

11.3.1 Cipher Suites

By default, the device is restricted to only support TLS Ciphersuites that are RFC compliant and can be claimed under MDFPP. As such, no configuration is needed to restrict or allow ciphersuites to be compliant. A list of the supported ciphersuites can be found below:

For TLS 1.2:

Approved Cipher Suites	TLS Version
TLS_RSA_WITH_AES_128_CBC_SHA256 as defined in RFC 5246	TLSv1.2
TLS_RSA_WITH_AES_256_CBC_SHA256 as defined in RFC 5246	
TLS_RSA_WITH_AES_256_GCM_SHA384 as defined in RFC 5288	
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 as defined in RFC 5289	
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 as defined in RFC 5289	
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 as defined in RFC 5289	
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 as defined in RFC 5289	
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 as defined in RFC 5289	
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 as defined in RFC 5289	
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 as defined in RFC 5289	
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 as defined in RFC 5289	

The device supports TLS versions 1.0, 1.1, and 1.2 for use with EAP-TLS as part of WPA2. The TOE supports the following ciphersuites for this:

TLS_RSA_WITH_AES_128_CBC_SHA as defined in RFC 5246
TLS_RSA_WITH_AES_256_CBC_SHA as defined in RFC 5246
TLS_RSA_WITH_AES_128_CBC_SHA256 as defined in RFC 5246
TLS_RSA_WITH_AES_256_CBC_SHA256 as defined in RFC 5246
TLS_RSA_WITH_AES_128_GCM_SHA256 as defined in RFC 5288
TLS_RSA_WITH_AES_256_GCM_SHA384 as defined in RFC 5288
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 as defined in RFC 5289

TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 as defined in RFC 5289
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 as defined in RFC 5289
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 as defined in RFC 5289
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 as defined in RFC 5289
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 as defined in RFC 5289
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 as defined in RFC 5289
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 as defined in RFC 5289

11.3.2 Guidance for Bluetooth Low Energy APIs

Provides classes that manage Bluetooth functionality, such as scanning for devices, connecting with devices, and managing data transfer between devices. The Bluetooth API supports both "Classic Bluetooth" and Bluetooth Low Energy.
For more information about Classic Bluetooth, see the Bluetooth guide. For more information about Bluetooth Low Energy, see the Bluetooth Low Energy (BLE) guide.
The Bluetooth APIs let applications:

- Scan for other Bluetooth devices (including BLE devices).
- Query the local Bluetooth adapter for paired Bluetooth devices.
- Establish RFCOMM channels/sockets.
- Connect to specified sockets on other devices.
- Transfer data to and from other devices.
- Communicate with BLE devices, such as proximity sensors, heart rate monitors, fitness devices, and so on.
- Act as a GATT client or a GATT server (BLE).

To perform Bluetooth communication using these APIs, an application must declare the BLUETOOTH permission. Some additional functionality, such as requesting device discovery, also requires the BLUETOOTH_ADMIN permission.

Interfaces

BluetoothAdapter.LeScanCallback	Callback interface used to deliver LE scan results
BluetoothProfile	Public APIs for the Bluetooth Profiles
BluetoothProfile.ServiceListener	An interface for notifying BluetoothProfile IPC clients when they have been connected or disconnected to the service

Classes

BluetoothA2dp	This class provides the public APIs to control the Bluetooth A2DP profile
BluetoothAdapter	Represents the local device Bluetooth adapter
BluetoothAssignedNumbers	Bluetooth Assigned Numbers

BluetoothClass	Represents a Bluetooth class, which describes general characteristics and capabilities of a device
BluetoothClass.Device	Defines all device class constants
BluetoothClass.Device.Major	Defines all major device class constants
BluetoothClass.Service	Defines all service class constants
BluetoothDevice	Represents a remote Bluetooth device
BluetoothGatt	Public API for the Bluetooth GATT Profile
BluetoothGattCallback	This abstract class is used to implement BluetoothGatt callbacks
BluetoothGattCharacteristic	Represents a Bluetooth GATT Characteristic A GATT characteristic is a basic data element used to construct a GATT service, BluetoothGattService
BluetoothGattDescriptor	Represents a Bluetooth GATT Descriptor GATT Descriptors contain additional information and attributes of a GATT characteristic, BluetoothGattCharacteristic
BluetoothGattServer	Public API for the Bluetooth GATT Profile server role
BluetoothGattServerCallback	This abstract class is used to implement BluetoothGattServer callbacks
BluetoothGattService	Represents a Bluetooth GATT Service Gatt Service contains a collection of BluetoothGattCharacteristic, as well as referenced services
BluetoothHeadset	Public API for controlling the Bluetooth Headset Service
BluetoothHealth	*This class was deprecated in API level 29. Health Device Profile (HDP) and MCAP protocol are no longer used. New apps should use Bluetooth Low Energy based solutions such as* BluetoothGatt, BluetoothAdapter#listenUsingL2capChannel(), *or* BluetoothDevice#createL2capChannel(int)
BluetoothHealthAppConfiguration	*This class was deprecated in API level 29. Health Device Profile (HDP) and MCAP protocol are no longer used. New apps should use Bluetooth Low Energy based solutions such as* BluetoothGatt, BluetoothAdapter#listenUsingL2capChannel(), *or* BluetoothDevice#createL2capChannel(int)

BluetoothHealthCallback	*This class was deprecated in API level 29. Health Device Profile (HDP) and MCAP protocol are no longer used. New apps should use Bluetooth Low Energy based solutions such as* BluetoothGatt,BluetoothAdapter#listenUsingL2capChannel(), *or* BluetoothDevice#createL2capChannel(int)
BluetoothHearingAid	This class provides the public APIs to control the Hearing Aid profile
BluetoothHidDevice	Provides the public APIs to control the Bluetooth HID Device profile
BluetoothHidDevice.Callback	The template class that applications use to call callback functions on events from the HID host
BluetoothHidDeviceAppQosSettings	Represents the Quality of Service (QoS) settings for a Bluetooth HID Device application
BluetoothHidDeviceAppSdpSettings	Represents the Service Discovery Protocol (SDP) settings for a Bluetooth HID Device application
BluetoothManager	High level manager used to obtain an instance of a BluetoothAdapter and to conduct overall Bluetooth Management
BluetoothServerSocket	A listening Bluetooth socket
BluetoothSocket	A connected or connecting Bluetooth socket

https://developer.android.com/reference/android/bluetooth/package-summary.html

How to connect and pair with a bluetooth device:

```
// get bluetooth adapter
BluetoothAdapter bluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
if (bluetoothAdapter == null) {
    // Device doesn't support Bluetooth
}

// make sure bluetooth is enabled
if (!bluetoothAdapter.isEnabled()) {
    Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
    startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);
}

// query for devices
Set<BluetoothDevice> pairedDevices = bluetoothAdapter.getBondedDevices();
if (pairedDevices.size() > 0) {
    // There are paired devices. Get the name and address of each paired device.
```

```
    for (BluetoothDevice device : pairedDevices) {
        String deviceName = device.getName();
        String deviceHardwareAddress = device.getAddress(); // MAC address
    }
}

// Connect to devices.
private class AcceptThread extends Thread {
    private final BluetoothServerSocket mmServerSocket;

    public AcceptThread() {
        // Use a temporary object that is later assigned to mmServerSocket
        // because mmServerSocket is final.
        BluetoothServerSocket tmp = null;
        try {
            // MY_UUID is the app's UUID string, also used by the client code.
            tmp = bluetoothAdapter.listenUsingRfcommWithServiceRecord(NAME, MY_UUID);
        } catch (IOException e) {
            Log.e(TAG, "Socket's listen() method failed", e);
        }
        mmServerSocket = tmp;
    }

    public void run() {
        BluetoothSocket socket = null;
        // Keep listening until exception occurs or a socket is returned.
        while (true) {
            try {
                socket = mmServerSocket.accept();
            } catch (IOException e) {
                Log.e(TAG, "Socket's accept() method failed", e);
                break;
            }

            if (socket != null) {
                // A connection was accepted. Perform work associated with
                // the connection in a separate thread.
                manageMyConnectedSocket(socket);
                mmServerSocket.close();
                break;
            }
        }
    }

    // Closes the connect socket and causes the thread to finish.
    public void cancel() {
        try {
```

```
            mmServerSocket.close();
        } catch (IOException e) {
            Log.e(TAG, "Could not close the connect socket", e);
        }
    }
}
```

More information here
https://developer.android.com/guide/topics/connectivity/bluetooth.html#SettingUp

Sample service to interact with a bluetooth APIs.

```
// A service that interacts with the BLE device via the Android BLE API.
public class BLEService extends Service {
    private final static String TAG = "BLEService";
    private BluetoothManager mBluetoothManager;
    private BluetoothAdapter mBluetoothAdapter;
    private String mBluetoothDeviceAddress;
    private BluetoothGatt mBluetoothGatt;
    private int mConnectionState = STATE_DISCONNECTED;
    private static final int STATE_DISCONNECTED = 0;
    private static final int STATE_CONNECTING = 1;
    private static final int STATE_CONNECTED = 2;
    public final static String ACTION_GATT_CONNECTED =
            "com.niap.ble.ACTION_GATT_CONNECTED";
    public final static String ACTION_GATT_DISCONNECTED =
            "com.niap.ble.ACTION_GATT_DISCONNECTED";
    public final static String ACTION_GATT_SERVICES_DISCOVERED =
            "com.niap.ble.ACTION_GATT_SERVICES_DISCOVERED";
    public final static String ACTION_DATA_AVAILABLE =
            "com.niap.ble.ACTION_DATA_AVAILABLE";
    public final static String EXTRA_DATA =
            "com.niap.ble.EXTRA_DATA";

    // Various callback methods defined by the BLE API.
    private final BluetoothGattCallback mGattCallback =
            new BluetoothGattCallback() {
                @Override
                public void onConnectionStateChange(BluetoothGatt gatt, int status,
                                                    int newState) {
                    String intentAction;
                    if (newState == BluetoothProfile.STATE_CONNECTED) {
                        intentAction = ACTION_GATT_CONNECTED;
                        mConnectionState = STATE_CONNECTED;
                        broadcastUpdate(intentAction);
                        Log.i(TAG, "Connected to GATT server.");
                        Log.i(TAG, "Attempting to start service discovery:" +
                                mBluetoothGatt.discoverServices());
```

```
            } else if (newState == BluetoothProfile.STATE_DISCONNECTED) {
                intentAction = ACTION_GATT_DISCONNECTED;
                mConnectionState = STATE_DISCONNECTED;
                Log.i(TAG, "Disconnected from GATT server.");
                broadcastUpdate(intentAction);
            }
        }

        @Override
        // New services discovered
        public void onServicesDiscovered(BluetoothGatt gatt, int status) {
            if (status == BluetoothGatt.GATT_SUCCESS) {
                broadcastUpdate(ACTION_GATT_SERVICES_DISCOVERED);
            } else {
                Log.w(TAG, "onServicesDiscovered received: " + status);
            }
        }

        @Override
        // Result of a characteristic read operation
        public void onCharacteristicRead(BluetoothGatt gatt,
                                         BluetoothGattCharacteristic characteristic,
                                         int status) {
            if (status == BluetoothGatt.GATT_SUCCESS) {
                broadcastUpdate(ACTION_DATA_AVAILABLE, characteristic);
            }
        }
    };
}
```

12. Version Information

This section provides instructions on how to check software, hardware, and application version information on Honeywell Devices.

12.1 Software Version via HUpgrader App

To check the currently installed build number on a device:

1. Tap on the HUpgrader application (pre-installed on all devices).
2. The device will inform the user of the latest image version installed.

12.2 Software and Hardware Versions via Settings App

To check hardware and software information of a device:

1. Tap on the “Settings” application
2. Select the “About phone” option

3. This menu shows basic information about the device, such as Android Version, Model, Build Number, and MAC/IP addresses. Some entries can be selected to provide further information, such as:
    a. Model & Hardware: Hardware Model and Serial Number
    b. Android Version: Android version, Security Patch Level, Kernel Version, and Build number
    c. Software Component Version: Contains various version numbers for different software components of the device

12.3 Application Version via Settings App

To check an application's version number:

1. Tap on the "Settings" application
2. Select the "Apps & Notifications" option
3. Tap "See all <number> apps"
4. Select the name of the application
5. Scroll down and tap on the "Advanced" portion
6. The version number will be at the very bottom of the extended menu