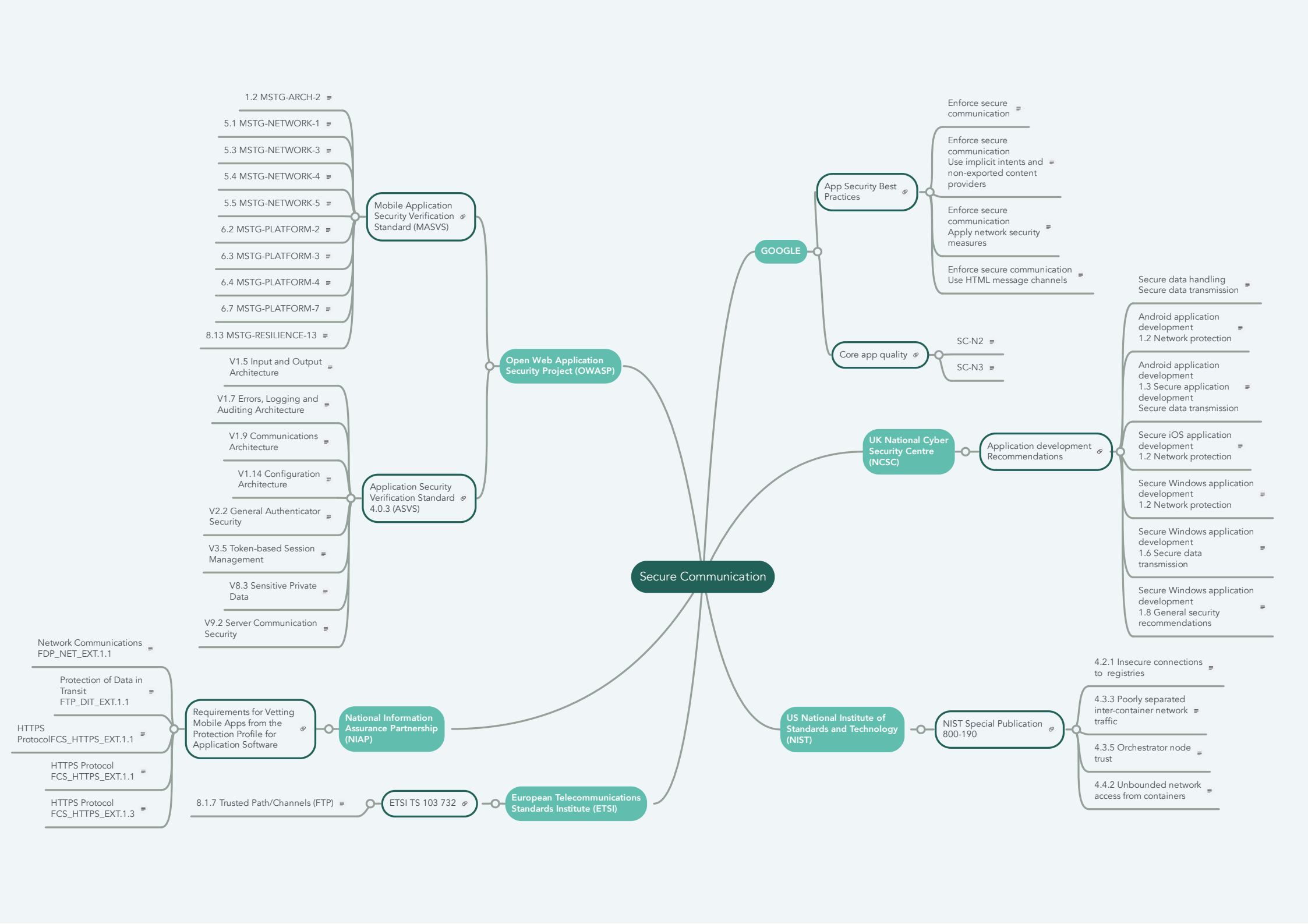
Secure Communication
Open Web Application Security Project (OWASP)
Mobile Application Security Verification Standard (MASVS)
1.2 MSTG-ARCH-2
5.1 MSTG-NETWORK-1
5.3 MSTG-NETWORK-3
5.4 MSTG-NETWORK-4
5.5 MSTG-NETWORK-5
6.2 MSTG-PLATFORM-2
6.3 MSTG-PLATFORM-3
6.4 MSTG-PLATFORM-4
6.7 MSTG-PLATFORM-7
8.13 MSTG-RESILIENCE-13
Application Security Verification Standard 4.0.3 (ASVS)
V1.5 Input and Output Architecture
V1.7 Errors, Logging and Auditing Architecture
V1.9 Communications Architecture
V1.14 Configuration Architecture
V2.2 General Authenticator Security
V3.5 Token-based Session Management
V8.3 Sensitive Private Data
V9.2 Server Communication Security
GOOGLE
App Security Best Practices
Enforce secure communication
Enforce secure communication Use implicit intents and non-exported content providers
Enforce secure communication Apply network security measures
Enforce secure communication Use HTML message channels
Core app quality
SC-N2
SC-N3
UK National Cyber Security Centre (NCSC)
Application development Recommendations
Secure data handling Secure data transmission
Android application development 1.2 Network protection
Android application development 1.3 Secure application development Secure data transmission
Secure iOS application development 1.2 Network protection
Secure Windows application development 1.2 Network protection
Secure Windows application development 1.6 Secure data transmission
Secure Windows application development 1.8 General security recommendations
US National Institute of Standards and Technology (NIST)
NIST Special Publication 800-190
4.2.1 Insecure connections to registries
4.3.3 Poorly separated inter-container network traffic
4.3.5 Orchestrator node trust
4.4.2 Unbounded network access from containers
National Information Assurance Partnership (NIAP)
Requirements for Vetting Mobile Apps from the Protection Profile for Application Software
Network Communications FDP_NET_EXT.1.1
Protection of Data in Transit FTP_DIT_EXT.1.1
HTTPS ProtocolFCS_HTTPS_EXT.1.1
HTTPS Protocol FCS_HTTPS_EXT.1.1
HTTPS Protocol FCS_HTTPS_EXT.1.3
European Telecommunications Standards Institute (ETSI)
ETSI TS 103 732
8.1.7 Trusted Path/Channels (FTP)

Secure Communication

1. GOOGLE

1.1. Core app quality

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

1.1.1. SC-N2

> The application declares a network security configuration.

1.1.2. SC-N3

> If the application uses Google Play services, the security provider is initialized at application startup.

1.2. App Security Best Practices

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

1.2.1. Enforce secure communication

> When you safeguard the data that you exchange between your app and other apps, or between your app and a website, you improve your app's stability and protect the data that you send and receive.

1.2.2. Enforce secure communication Use implicit intents and non-exported content providers

Show an app chooser
If an implicit intent can launch at least two possible apps on a user's device, explicitly show an app chooser. This interaction strategy allows users to transfer sensitive information to an app that they trust.
Apply signature-based permissions
When sharing data between two apps that you control or own, use signature-based permissions. These permissions don't require user confirmation and instead check that the apps accessing the data are signed using the same signing key. Therefore, these permissions offer a more streamlined, secure user experience.
Disallow access to your app's content providers
Unless you intend to send data from your app to a different app that you don't own, you should explicitly disallow other developers' apps from accessing the ContentProvider objects that your app contains. This setting is particularly important if your app can be installed on devices running Android 4.1.1 (API level 16) or lower, as the android:exported attribute of the element is true by default on those versions of Android.

1.2.3. Enforce secure communication Apply network security measures

Use TLS traffic
If your app communicates with a web server that has a certificate issued by awell-known, trusted certifi cate authority (CA), use an HTTPS request like the following:
Add a network security configuration
If your app uses new or custom CAs, you can declare your network's securitysettings in a confi guration fi le. This process lets you create the confi gurationwithout modifying any app code.
To add a network security confi guration fi le to your app, follow these steps:
1. Declare the confi guration in your app's manifest:
2. Add an XML resource fi le, located at res/xml/network_security_config.xml.
Specify that all traffi c to particular domains must use HTTPS by disablingclear-text:
During the development process, you can use the element to explicitly allow user-installed certifi cates. This elementoverrides your app's security-critical options during debugging and testingwithout affecting the app's release confi guration. The following snippetshows how to defi ne this element in your app's network securityconfi guration XML file:
Create your own trust manager
Your TLS checker shouldn't accept every certifi cate. You might need to set up atrust manager and handle all TLS warnings that occur if one of the followingconditions applies to your use case:
You're communicating with a web server that has a certifi cate signed by anew or custom CA.
That CA isn't trusted by the device you're using.

1.2.4. Enforce secure communication Use HTML message channels

> If your app must use JavaScript interface support on devices running Android6.0 (API level 23) and higher, use HTML message channels instead of communicating between a website and your app, as shown in the following code snippet:

2. UK National Cyber Security Centre (NCSC)

2.1. Application development Recommendations

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

2.1.1. Secure data handling Secure data transmission

> Send any sensitive information transferred between device and server using an appropriate encryption mechanism. All modern platforms have built-in support for transport layer security (TLS), which is the NCSC's preferred option.
> Restrict supported ciphers on both ends of the communication, so that only strong ciphers may be used. Take additional steps to maximise the security of the data connection, such as using certificate pinning to ensure the application connects to a host with a known, trusted certificate.
> Never send sensitive data over an insecure or unencrypted connection, and where possible non-sensitive data should also be sent over a secure connection.
> Alert the user if any suspicious attributes are detected that indicate the secure communication channel is under attack. In this case, the connection should be denied until a verified secure channel is available.

2.1.2. Android application development 1.2 Network protection

> The diagram below, taken from the EUD Security Guidance for Android, illustrates the recommended network configuration for Android devices which handle sensitive information. In summary, a VPN is used to bring device traffic back to the enterprise. Access to internal services is brokered through a reverse proxy server, which protects the internal network from attack.
>
> To prevent the application from accessing sensitive internal resources, it is important that the reverse proxy server authenticates any requests from devices. This means that applications on the device which are trusted to access sensitive data must provide authentication with each request so that the reverse proxy can validate the request. Stored credentials must be private to only the trusted applications accessing those resources.
>
> Internet requests from the application should be routed via the standard corporate internet gateway, to permit traffic inspection.

2.1.3. Android application development 1.3 Secure application development Secure data transmission

In order to transmit sensitive data securely, Android applications should conform to the following:

All off-device communications handling sensitive data should take place over a mutually-authenticated, cryptographically protected, connection.
For TLS connections, the application should perform certificate pinning to a known endpoint. This process should leverage the Network Security Configuration. For more information refer to the NCSC's TLS documentation.
Certificates used by the application should be stored on the device using the Android KeyStore provider.
Note that at present there is no API on Android to check the status of the VPN. To securely check the status of the VPN, the internal service with which the application is communicating must be authenticated. The recommended way of performing this authentication is TLS with a pinned certificate. If mutual authentication is required to the internal service, mutual TLS with pinned certificates should be used.

2.1.4. Secure iOS application development 1.2 Network protection

The diagram below, taken from the NCSC EUD Security Guidance for iOS, illustrates the recommended network configuration for iOS devices which handle sensitive information. In summary, a VPN is used to bring device traffic back to the enterprise. Access to internal services is brokered through a reverse proxy server, which protects the internal network from attack.

To prevent the application from accessing sensitive internal resources, it is important that the reverse proxy server authenticates any requests from devices. This means that applications on the device, which are trusted to access sensitive data, should provide authentication with each request so that the reverse proxy can validate the request. Stored credentials should be private to only the trusted applications accessing those resources.

Internet requests from the application should be routed via the standard corporate Internet gateway, to permit traffic inspection.

2.1.5. Secure Windows application development 1.2 Network protection

The diagram below, taken from the Windows 10 security guidance, illustrates the recommended network configuration for UWP devices handling sensitive data. In summary, a VPN is used to bring device traffic back to the enterprise. Access to internal services is brokered through a reverse proxy server which protects the internal network from attack.

2.1.6. Secure Windows application development 1.6 Secure data transmission

In order to transmit sensitive data securely, Windows applications should conform to the following:

For TLS connections, certificate pinning to known organisation services should be enforced.

Encrypt any Websocket connections using the wss: URI scheme.
All HTTP or HTTPS connections should use the Web.Http API.
Note that because UWP applications run in a contained environment, issuer certificates to be used for validation are stored in an isolated cache within the container; as a result, you do not need to do anything to store issuer certificates securely.

2.1.7. Secure Windows application development 1.8 General security recommendations

The following additional behaviours can increase the overall security of an application:

Applications should minimise the amount of sensitive data stored on the device, retrieving data from the server when needed over a secure connection, and erase it when it is no longer required.
Applications that require authentication on application launch should also request this authentication credential when returning back into the foreground after previously being backgrounded by a user allowing for a small grace period.

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

3.1. NIST Special Publication 800-190

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

3.1.1. 4.2.1 Insecure connections to registries

Organizations should configure their development tools, orchestrators, and container runtimes to only connect to registries over encrypted channels. The specific steps vary between tools, but the key goal is to ensure that all data pushed to and pulled from a registry occurs between trusted endpoints and is encrypted in transit.

3.1.2. 4.3.3 Poorly separated inter-container network traffic

Orchestrators should be configured to separate network traffic into discrete virtual networks by sensitivity level. While per-app segmentation is also possible, for most organizations and use cases, simply defining networks by sensitivity level provides sufficient mitigation of risk with a manageable degree of complexity. For example, public-facing apps can share a virtual network, internal apps can use another, and communication between the two should occur through a small number of well-defined interfaces.

3.1.3. 4.3.5 Orchestrator node trust

Orchestration platforms should be configured to provide features that create a secure environment for all the apps they run. Orchestrators should ensure that nodes are securely introduced to the cluster, have a persistent identity throughout their lifecycle, and can also provide an accurate inventory of nodes and their connectivity states. Organizations should ensure that orchestration platforms are designed specifically to be resilient to compromise of individual nodes without compromising the overall security of the cluster. A compromised node must be able to be isolated and removed from the cluster without disrupting or degrading overall cluster operations. Finally, organizations should choose orchestrators that provide mutually authenticated network connections between cluster members and end-to-end encryption of intracluster traffic. Because of the portability of containers, many deployments may occur across networks organizations do not directly control, so a secure-by-default posture is particularly important for this scenario.

3.1.4. 4.4.2 Unbounded network access from containers

Organizations should control the egress network traffic sent by containers. At minimum, these controls should be in place at network borders, ensuring containers are not able to send traffic across networks of differing sensitivity levels, such as from an environment hosting secure data to the internet, similar to the patterns used for traditional architectures. However, the virtualized networking model of inter-container traffic poses an additional challenge.
Because containers deployed across multiple hosts typically communicate over a virtual, encrypted network, traditional network devices are often blind to this traffic. Additionally, containers are typically assigned dynamic IP addresses automatically when deployed by orchestrators, and these addresses change continuously as the app is scaled and load balanced. Thus, ideally, organizations should use a combination of existing network level devices and more app-aware network filtering. App-aware tools should be able to not just see the intercontainer traffic, but also to dynamically generate the rules used to filter this traffic based on the specific characteristics of the apps running in the containers. This dynamic rule management is
critical due to the scale and rate of change of containerized apps, as well as their ephemeral networking topology.
Specifically, app-aware tools should provide the following capabilities:
• Automated determination of proper container networking surfaces, including both inbound ports and process-port bindings;
• Detection of traffic flows both between containers and other network entities, over both ‘on the wire’ traffic and encapsulated traffic; and
• Detection of network anomalies, such as unexpected traffic flows within the organization’s network, port scanning, or outbound access to potentially dangerous destinations.

4. European Telecommunications Standards Institute (ETSI)

4.1. ETSI TS 103 732

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

4.1.1. 8.1.7 Trusted Path/Channels (FTP)

A TOE usually supports many different communication channels, conforming to different standards, and within these standards, using different settings, resulting in different levels of security.
The ST author shall provide FTP_ITC SFRs for each channel that the ST author claims to be secure. At least one secure channel shall be claimed in the Security Target.
This does not preclude the TOE providing communication channels based on these standards with less secure settings to communicate with devices that are legacy or have limited secure communication capabilities. As part of the vulnerability analysis the impact these lower settings can have on the overall security of the TOE will be assessed.
The TOE should provide the ability for downloaded apps to use some of the secure channel mechanisms for their communications. In this case, the "trusted IT product" that the app communicated with will be determined by the app itself.
FTP_ITC.1 Inter-TSF trusted channel
FTP_ITC.1.1 The TSF shall 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.1.2 The TSF shall permit [selection: the TSF, another trusted IT product] to initiate communication via the trusted channel.
FTP_ITC.1.3 The TSF shall initiate communication via the trusted channel for communication with the trusted IT product.
If Bluetooth is supported by the TOE, the Bluetooth channel shall as a minimum:
• conform to Bluetooth® Core Specification selection v4.1 [12], v4.2 [13], v5.0 [14], v5.1 [15], v5.2 [16] or higher version of Bluetooth Core Specification; and
• 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 [selection: 24 hours, three failed authentication attempts from any Bluetooth device address, ten successful pairings from any Bluetooth device address, [assignment: other frequency and/or criteria for new key pair generation]].
If HTTPs is supported by the TOE, the HTTPs channel shall as a minimum:
• conform to IETF RFC 2818 [6]; and
• use TLS v1.2 [7], TLS v1.3 [11], or higher version of TLS to implement HTTPs.
If TLS is supported by the TOE, the TLS channel shall as a minimum:
• implement TLS v1.2 [7], TLS v1.3 [11] or higher version of TLS; and
• 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 [8]; 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
• support one of the following ciphersuites:
- TLS_RSA_WITH_AES_256_GCM_SHA384 (IETF RFC 5288 [9])
- TLS_DHE_RSA_WITH_AES_256_GCM_SHA384 (IETF RFC 5288 [9])
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 (IETF RFC 5289 [10])
- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (in IETF RFC 5289 [10])
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 (in IETF RFC 5289 [10])
- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (in IETF RFC 5289 [10])
- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 (IETF RFC 5289 [10])
- TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 (IETF RFC 5289 [10])
- TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (IETF RFC 5289 [10])
- TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (IETF RFC 5289 [10])
If WLAN is supported by the TOE, the WLAN channel shall as a minimum:
• implement 802.11-2012 [17], 802.1X [18] and EAP-TLS [19]; and
• generate symmetric keys according to PRF 384 or PRF-704 with key length 128 bit or 256 bit; and
• use TLS v1.2 [7] or higher; and
• 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 [8]; and
• notify the TSF, and not establish the connection or request application authorization to establish the connection if the peer certificate is deemed invalid; and
• support one of the following ciphersuites:
- TLS_RSA_WITH_AES_256_GCM_SHA384 (IETF RFC 5288 [9])
- TLS_DHE_RSA_WITH_AES_256_GCM_SHA384 (IETF RFC 5288 [9])
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 (IETF RFC 5289 [10])
- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (IETF RFC 5289 [10])
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 (IETF RFC 5289 [10])
- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (IETF RFC 5289 [10])
- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 (IETF RFC 5289 [10])
- TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 (IETF RFC 5289 [10])
- TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (IETF RFC 5289 [10])

- TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (IETF RFC 5289 [10])
• Randomly generate a new MAC-address each time it connects to a different access point.

5. National Information Assurance Partnership (NIAP)

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

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

5.1.1. Network Communications FDP_NET_EXT.1.1

The application shall restrict network communication to [selection:
no network communication,

user-initiated communication for [assignment: list of functions for which the user can initiate network communication] ,respond to [assignment: list of remotely initiated communication] ,[assignment: list of application-initiated network communication]] .
Application Note: This requirement is intended to restrict both inbound and outbound network communications to only those required, or to network communications that are user initiated. It does not apply to network communications in which the application may generically access the filesystem which may result in the platform accessing remotely mounted drives/shares.

5.1.2. Protection of Data in Transit FTP_DIT_EXT.1.1

The TSF shall be capable of performing the following management functions [selection:
no management functions,

enable/disable the transmission of any information describing the system's hardware, software, or configuration ,

enable/disable the transmission of any PII ,

enable/disable transmission of any application state (e.g. crashdump) information,

enable/disable network backup functionality to [assignment: list of enterprise or commercial cloud backup systems] ,

[assignment: list of other management functions to be provided by the TSF]

] .
Application Note: this requirement stipulates that an application needs to provide the ability to enable/disable only those functions that it actually implements. the application is not responsible for controlling the behavior of the platform or other applications.

5.1.3. HTTPS ProtocolFCS_HTTPS_EXT.1.1

The application shall [selection:
not transmit any data,

not transmit any sensitive data,

encrypt all transmitted sensitive data with [selection, at least one of: HTTPS, TLS, DTLS, SSH as conforming to the Extended Package for Secure Shell] ,

encrypt all transmitted data with [selection, at least one of: HTTPS, TLS, DTLS, SSH]

] between itself and another trusted IT product.
Application Note: Extended packages may override this requirement to provide for other protocols. Encryption is not required for applications transmitting data that is not sensitive.
If TLS is selected, then evaluation of elements from FCS_TLSC_EXT.1 is required.

If HTTPS is selected, then evaluation of elements from FCS_HTTPS_EXT.1 is required.

If DTLS is selected, then evaluation of elements from FCS_DTLS_EXT.1 is required.

If SSH is selected, the TSF shall be validated against the Extended Package for Secure Shell.

5.1.4. HTTPS Protocol FCS_HTTPS_EXT.1.1

The application shall implement the HTTPS protocol that complies with RFC 2818.

5.1.5. HTTPS Protocol FCS_HTTPS_EXT.1.3

The application shall notify the user and [selection: not establish the connection , request application authorization to establish the connection , no other action] if the peer certificate is deemed invalid.
Application Note: Validity is determined by the certificate path, the expiration date, and the revocation status in accordance with RFC 5280.

6. Open Web Application Security Project (OWASP)

6.1. Mobile Application Security Verification Standard (MASVS)

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

6.1.1. 1.2 MSTG-ARCH-2

> Security controls are never enforced only on the client side, but on the respective remote endpoints.

6.1.2. 5.1 MSTG-NETWORK-1

> Data is encrypted on the network using TLS. The secure channel is used consistently throughout the app.

6.1.3. 5.3 MSTG-NETWORK-3

> The app verifies the X.509 certificate of the remote endpoint when the secure channel is established. Only certificates signed by a trusted CA are accepted.

6.1.4. 5.4 MSTG-NETWORK-4

> The app either uses its own certificate store, or pins the endpoint certificate or public key, and subsequently does not establish connections with endpoints that offer a different certificate or key, even if signed by a trusted CA.

6.1.5. 5.5 MSTG-NETWORK-5

> The app doesn't rely on a single insecure communication channel (email or SMS) for critical operations, such as enrollments and account recovery.

6.1.6. 6.2 MSTG-PLATFORM-2

> 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.

6.1.7. 6.3 MSTG-PLATFORM-3

> The app does not export sensitive functionality via custom URL schemes, unless these mechanisms are properly protected.

6.1.8. 6.4 MSTG-PLATFORM-4

The app does not export sensitive functionality through IPC facilities, unless these mechanisms are properly protected.

6.1.9. 6.7 MSTG-PLATFORM-7

If native methods of the app are exposed to WebView, verify that the WebView only renders JavaScript contained within the app package

6.1.10. 8.13 MSTG-RESILIENCE-13

As a defense in depth, next to having solid hardening of the communicating parties, application level payload encryption can be applied to further impede eavesdropping.

6.2. Application Security Verification Standard 4.0.3 (ASVS)

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

6.2.1. V1.5 Input and Output Architecture

1.5.2 Verify that serialization is not used when communicating with untrusted clients. If this is not possible, ensure that adequate integrity controls (and possibly encryption if sensitive data is sent) are enforced to prevent deserialization attacks including object injection.

6.2.2. V1.7 Errors, Logging and Auditing Architecture

1.7.2 Verify that logs are securely transmitted to a preferably remote system for analysis, detection, alerting, and escalation.

6.2.3. V1.9 Communications Architecture

1.9.1 Verify the application encrypts communications between components, particularly when these components are in different containers, systems, sites, or cloud providers.

1.9.2 Verify that application components verify the authenticity of each side in a communication link to prevent person-in-the-middle attacks. For example, application components should validate TLS certificates and chains.

6.2.4. V1.14 Configuration Architecture

1.14.2 Verify that binary signatures, trusted connections, and verified endpoints are used to deploy binaries to remote devices

6.2.5. V2.2 General Authenticator Security

2.2.5 Verify that where a Credential Service Provider (CSP) and the application verifying authentication are separated, mutually authenticated TLS is in place between the two endpoints.

2.2.6 Verify replay resistance through the mandated use of One-time Passwords (OTP) devices, cryptographic authenticators, or lookup codes.

6.2.6. V3.5 Token-based Session Management

3.5.3 Verify that stateless session tokens use digital signatures, encryption, and other countermeasures to protect against tampering, enveloping, replay, null cipher, and key substitution attacks.

6.2.7. V8.3 Sensitive Private Data

8.3.1 Verify that sensitive data is sent to the server in the HTTP message body or headers, and that query string parameters from any HTTP verb do not contain sensitive data.

6.2.8. V9.2 Server Communication Security

9.2.2 Verify that encrypted communications such as TLS is used for all inbound and outbound connections, including for management ports, monitoring, authentication, API, or web service calls, database, cloud, serverless, mainframe, external, and partner connections. The server must not fall back to insecure or unencrypted protocols.

9.2.3 Verify that all encrypted connections to external systems that involve sensitive information or functions are authenticated.