

# Passwords and Authentication

## National Information Assurance Partnership (NIAP)

### Requirements for Vetting Mobile Apps from the Protection Profile for Application Software

- Secure by Default Configuration FMT\_CFG\_EXT.1.1
- Password Conditioning FCS\_CKM.1(B)

## US National Institute of Standards and Technology (NIST)

### NIST Special Publication 800-190

- 4.2.3 Insufficient authentication and authorization restrictions
- 4.3.2 Unauthorized access

## UK National Cyber Security Centre (NCSC)

### Application development Recommendations

- Android application development
  - 1.3 Secure application development
  - Server-side controls
- Secure iOS application development
  - 1.3 Secure application development recommendations
  - Server-side controls
- Secure Windows application development
  - 1.3 Authentication
- Secure Windows application development
  - 1.5 Server-side controls

## European Telecommunications Standards Institute (ETSI)

### ETSI TS 103 732

- 8.1.1 Cryptographic Support (FCS) FCS\_RNG.1 Random number generation
- 8.1.3 Identification and Authentication (FIA) FIA\_UAU.1 Timing of authentication
- 8.1.3 Identification and Authentication (FIA) FIA\_UAU.2 User authentication before any action
- 8.1.3 Identification and Authentication (FIA) FIA\_UAU.5 Multiple authentication mechanisms
- 8.1.3 Identification and Authentication (FIA) FIA\_UAU.6 Re-authenticating
- 8.1.3 Identification and Authentication (FIA) FIA\_SOS.1 Verification of secrets
- 8.1.3 Identification and Authentication (FIA) FIA\_UAU.7 Protected authentication feedback
- 8.1.3 Identification and Authentication (FIA) FIA\_AFL.1 Other Authentication failure handling
- 8.1.4 Security Management (FMT) FMT\_SMF.1 Authentication Specification of Management Functions
- 8.1.3 Identification and Authentication (FIA) FIA\_AFL.1 Password/PIN/Pattern Authentication failure handling
- 8.1.3 Identification and Authentication (FIA) FIA\_SOS.2 TSF Generation of secrets

## APPLE

### Developer Security

- Authorization and Authentication Password AutoFill Overview
- Authorization and Authentication Password AutoFill Enable Password AutoFill
- Authorization and Authentication Password AutoFill Support Third-Party Web Services
- Authorization and Authentication Password AutoFill Integrate a Password Management App with Password AutoFill
- Authorization and Authentication Shared Web Credentials Overview
- Authorization and Authentication Authorization Services Overview
- Authorization and Authentication Authorization Plug-ins Overview

### Mobile Application Profile

- 4.4. No Universal Password UP1
- 4.4. No Universal Password UP103
- 4.4. No Universal Password UP104
- 4.4. No Universal Password UP105
- 4.4. No Universal Password UP2.1
- 4.4. No Universal Password UP2.2
- 4.4. No Universal Password UP107

## IoT Alliance

- 2.7 MSG-STORAGE-7
- 2.11 MSG-STORAGE-11
- 4.1 MSG-AUTH-1
- 4.2 MSG-AUTH-2
- 4.3 MSG-AUTH-3
- 4.5 MSG-AUTH-5
- 4.6 MSG-AUTH-6
- 4.8 MSG-AUTH-8
- 4.9 MSG-AUTH-9
- 4.10 MSG-AUTH-10

## Open Web Application Security Project (OWASP)

### Mobile Application Security Verification Standard (MASVS)

- V1.2 Authentication Architecture
- V1.4 Access Control Architecture
- V1.6 Cryptographic Architecture
- V2.1 Password Security
- V2.2 General Authenticator Security
- V2.3 Authenticator Lifecycle
- V2.5 Credential Recovery
- V2.7 Out of Band Verifier
- V2.8 One Time Verifier
- V2.10 Service Authentication
- V3.5 Token-based Session Management
- V3.6 Federated Re-authentication
- V3.7 Defenses Against Session Management Exploits
- V4.3 Other Access Control Considerations
- V14.5 HTTP Request Header Validation

## GOOGLE

### Core app quality

- SC-ID1
- SC-ID2
- SC-ID3

### App Security Best Practices

- Enforce secure communication
- Ask for credentials before showing sensitive information

# Passwords and Authentication

## 1. APPLE

### 1.1. Developer Security

**Link:** <https://developer.apple.com/documentation/security>

#### 1.1.1. Authorization and Authentication Password AutoFill Overview

Password AutoFill simplifies login and account creation tasks for iOS apps and webpages. With just a few taps, your users can create and save new passwords or log in to an existing account. Users don't even need to know their password; the system handles everything. This convenience increases the likelihood that users will complete your app's onboarding process and start using your app more quickly. Additionally, by encouraging users to select unique, strong passwords, you increase the security of your app.

By default, Password AutoFill saves the user's login credentials on their current iOS device. iOS can sync these credentials securely across the user's devices using iCloud Keychain. Password AutoFill recommends credentials only for your app's associated domain, and the user must authenticate using Face ID or Touch ID before accessing these credentials. For more information on privacy and security, see [Approach to Privacy and iOS Security Guide](#).

Password AutoFill also provides credentials from third-party password managers that implement a credential provider extension. For more information on the credential provider extension, see the [Authentication Services framework](#).

#### 1.1.2. Authorization and Authentication Password AutoFill Enable Password AutoFill

Password AutoFill uses heuristics to determine when the user logs in or creates new passwords, and automatically provides the password QuickType bar. These heuristics give users some Password AutoFill support in most apps, even if those apps haven't been updated to support AutoFill. However, to provide the best user experience and ensure your app fully supports Password AutoFill, perform the following steps:

Set up your app's associated domains. To learn how to set up your app's associated domains, see [Supporting Associated Domains](#).

Set the correct AutoFill type on relevant text fields. For an iOS app, see [Enabling Password AutoFill on a Text Input View](#). For a web app, see [Enabling Password AutoFill on an HTML Input Element](#).

#### 1.1.3. Authorization and Authentication Password AutoFill Support Third-Party Web Services

Password AutoFill streamlines logging into web services at your domain; however, if you need to log into a third-party service, use `ASWebAuthenticationSession` instead, which supports Password AutoFill when your user has not already logged in.

#### 1.1.4. Authorization and Authentication Password AutoFill Integrate a Password Management App with Password AutoFill

If you are developing a password management app, create AutoFill Credential Provider Extensions to surface credentials from your app in Password Autofill and pull your app's password data into the Password AutoFill workflow. When your app integrates with Password AutoFill, users don't have to copy and paste credentials. Instead, they can use password data stored in your app easily because the data will be offered to the user to fill in compatible user name and password fields. To integrate a password app with Password AutoFill, use in the Authentication Services framework.

#### 1.1.5. Authorization and Authentication Shared Web Credentials Overview

The `Security.SecSharedCredentials` API provides functions for storing and requesting shared password-based credentials. Users often save their username and password in their iCloud keychain when logging into websites in Safari. Later, they may run a native app from the same developer to access the same account. With shared web credentials, the app can access the credentials stored for the website instead of requiring the user to reenter a username and password. Users can also create new accounts, update passwords, or delete their accounts from within the app. These changes are then saved and used by Safari.

#### 1.1.6. Authorization and Authentication Authorization Services Overview

The Security.Authorization API is a programming interface to the Security Server and its policy database. This API facilitates access control to restricted areas of the operating system and allows you to restrict a user's access to particular features in your macOS app. Use authorization services in:

Software that restricts access to its own tools

Applications that call system tools

Software installers that install privileged tools or require access to restricted areas of the operating system

As shown in Figure 1, the Security Server is a daemon running in the operating system that provides a trusted implementation of various security protocols, including authorization computation. In turn, the Security Server relies on the Security Agent to interface with users when authentication is needed. Thus an app can verify credentials (usernames and passwords) without ever accessing them directly. This authorization process also allows the means of authentication to change in the future (such as adding Touch ID) without your having to modify your app.

#### 1.1.7. Authorization and Authentication Authorization Plug-ins Overview

Use plug-ins to extend macOS authorization services to perform authorizations in a new way or to implement a new policy that is too complex to be implemented entirely with the authorization policy database.

You must import this API explicitly:

```
import Security.AuthorizationPlugin
```

## 2. ioXt Alliance

### 2.1. Mobile Application Profile

**Link:** [https://static1.squarespace.com/static/5c6dbac1f8135a29c7fbb621/t/604aa3fa668a8e3b50630433/1615504379349/Mobile\\_Application\\_Profile.pdf](https://static1.squarespace.com/static/5c6dbac1f8135a29c7fbb621/t/604aa3fa668a8e3b50630433/1615504379349/Mobile_Application_Profile.pdf)

#### 2.1.1. 4.4. No Universal Password UP1

User credentials shall not be common or predictable, or the credentials must be required to change at initial use.

#### 2.1.2. 4.4. No Universal Password UP103

Require authentication for remote services containing user data.

#### 2.1.3. 4.4. No Universal Password UP104

Enforce a strong server-side password policy.

#### 2.1.4. 4.4. No Universal Password UP105

Limit lifetime of authentication materials.

#### 2.1.5. 4.4. No Universal Password UP2.1

Availability of two factor authentication for products which have a user facing interface during initialization

#### 2.1.6. 4.4. No Universal Password UP2.2

Availability of two factor authentication for products which have a user facing interface during management

#### 2.1.7. 4.4. No Universal Password UP107

App shall re-authenticate the user when displaying sensitive PII data or conducting sensitive transactions.

### 3. Open Web Application Security Project (OWASP)

#### 3.1. Mobile Application Security Verification Standard (MASVS)

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

##### 3.1.1. 2.7 MSTG-STORAGE-7

No sensitive data, such as passwords or pins, is exposed through the user interface

##### 3.1.2. 2.11 MSTG-STORAGE-11

The app enforces a minimum device-access-security policy, such as requiring the user to set a device passcode.

#### 3.1.3. 4.1 MSTG-AUTH-1

If the app provides users access to a remote service, some form of authentication, such as username/password authentication, is performed at the remote endpoint.

#### 3.1.4. 4.2 MSTG-AUTH-2

If stateful session management is used, the remote endpoint uses randomly generated session identifiers to authenticate client requests without sending the user's credentials.

#### 3.1.5. 4.3 MSTG-AUTH-3

If stateless token-based authentication is used, the server provides a token that has been signed using a secure algorithm.

#### 3.1.6. 4.5 MSTG-AUTH-5

A password policy exists and is enforced at the remote endpoint.

#### 3.1.7. 4.6 MSTG-AUTH-6

The remote endpoint implements a mechanism to protect against the submission of credentials an excessive number of times.

#### 3.1.8. 4.8 MSTG-AUTH-8

Biometric authentication, if any, is not event-bound (i.e. using an API that simply returns "true" or "false"). Instead, it is based on unlocking the keychain/keystore.

#### 3.1.9. 4.9 MSTG-AUTH-9

A second factor of authentication exists at the remote endpoint and the 2FA requirement is consistently enforced.

#### 3.1.10. 4.10 MSTG-AUTH-10

Sensitive transactions require step-up authentication

### 3.2. 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>

#### 3.2.1. V1.2 Authentication Architecture

1.2.3 Verify that the application uses a single vetted authentication mechanism that is known to be secure, can be extended to include strong authentication, and has sufficient logging and monitoring to detect account abuse or breaches.

1.2.4 Verify that all authentication pathways and identity management APIs implement consistent authentication security control strength, such that there are no weaker alternatives per the risk of the application.

#### 3.2.2. V1.4 Access Control Architecture

1.4.1 Verify that trusted enforcement points, such as access control gateways, servers, and serverless functions, enforce access controls. Never enforce access controls on the client.

1.4.4 Verify the application uses a single and well-vetted access control mechanism for accessing protected data and resources. All requests must pass through this single mechanism to avoid copy and paste or insecure alternative paths.

1.4.5 Verify that attribute or feature-based access control is used whereby the code checks the user's authorization for a feature/data item rather than just their role. Permissions should still be allocated using roles.

#### 3.2.3. V1.6 Cryptographic Architecture

1.6.3 Verify that all keys and passwords are replaceable and are part of a well-defined process to re-encrypt sensitive data.

#### 3.2.4. V2.1 Password Security

2.1.1 Verify that user set passwords are at least 12 characters in length (after multiple spaces are combined).

2.1.2 Verify that passwords of at least 64 characters are permitted, and that passwords of more than 128 characters are denied.

2.1.3 Verify that password truncation is not performed. However, consecutive multiple spaces may be replaced by a single space.

2.1.4 Verify that any printable Unicode character, including language neutral characters such as spaces and Emojis are permitted in passwords.

2.1.5 Verify users can change their password.

2.1.6 Verify that password change functionality requires the user's current and new password.

2.1.7 Verify that passwords submitted during account registration, login, and password change are checked against a set of breached passwords either locally (such as the top 1,000 or 10,000 most common passwords which match the system's password policy) or using an external API. If using an API a zero knowledge proof or other mechanism should be used to ensure that the plain text password is not sent or used in verifying the breach status of the password. If the password is breached, the application must require the user to set a new non-breached password.

2.1.8 Verify that a password strength meter is provided to help users set a stronger password.

2.1.9 Verify that there are no password composition rules limiting the type of characters permitted. There should be no requirement for upper or lower case or numbers or special characters.

2.1.10 Verify that there are no periodic credential rotation or password history requirements.

2.1.11 Verify that "paste" functionality, browser password helpers, and external password managers are permitted.

2.1.12 Verify that the user can choose to either temporarily view the entire masked password, or temporarily view the last typed character of the password on platforms that do not have this as built-in functionality.

### 3.2.5. V2.2 General Authenticator Security



2.2.1 Verify that anti-automation controls are effective at mitigating breached credential testing, brute force, and account lockout attacks. Such controls include blocking the most common breached passwords, soft lockouts, rate limiting, CAPTCHA, ever increasing delays between attempts, IP address restrictions, or risk-based restrictions such as location, first login on a device, recent attempts to unlock the account, or similar. Verify that no more than 100 failed attempts per hour is possible on a single account.

2.2.2 Verify that the use of weak authenticators (such as SMS and email) is limited to secondary verification and transaction approval and not as a replacement for more secure authentication methods. Verify that stronger methods are offered before weak methods, users are aware of the risks, or that proper measures are in place to limit the risks of account compromise.

2.2.3 Verify that secure notifications are sent to users after updates to authentication details, such as credential resets, email or address changes, logging in from unknown or risky locations. The use of push notifications - rather than SMS or email - is preferred, but in the absence of push notifications, SMS or email is acceptable as long as no sensitive information is disclosed in the notification.

2.2.4 Verify impersonation resistance against phishing, such as the use of multi-factor authentication, cryptographic devices with intent (such as connected keys with a push to authenticate), or at higher AAL levels, client-side certificates.

2.2.7 Verify intent to authenticate by requiring the entry of an OTP token or user-initiated action such as a button press on a FIDO hardware key.

### 3.2.6. V2.3 Authenticator Lifecycle

2.3.1 Verify system generated initial passwords or activation codes SHOULD be securely randomly generated, SHOULD be at least 6 characters long, and MAY contain letters and numbers, and expire after a short period of time. These initial secrets must not be permitted to become the long term password.

2.3.2 Verify that enrollment and use of user-provided authentication devices are supported, such as a U2F or FIDO tokens.

### 3.2.7. V2.5 Credential Recovery

2.5.5 Verify that if an authentication factor is changed or replaced, that the user is notified of this event.

2.5.6 Verify forgotten password, and other recovery paths use a secure recovery mechanism, such as time-based OTP (TOTP) or other soft token, mobile push, or another offline recovery mechanism.

2.5.7 Verify that if OTP or multi-factor authentication factors are lost, that evidence of identity proofing is performed at the same level as during enrollment.

### 3.2.8. V2.7 Out of Band Verifier

2.7.1 Verify that clear text out of band (NIST "restricted") authenticators, such as SMS or PSTN, are not offered by default, and stronger alternatives such as push notifications are offered first.

2.7.2 Verify that the out of band verifier expires out of band authentication requests, codes, or tokens after 10 minutes.

2.7.3 Verify that the out of band verifier authentication requests, codes, or tokens are only usable once, and only for the original authentication request.

2.7.4 Verify that the out of band authenticator and verifier communicates over a secure independent channel.

2.7.5 Verify that the out of band verifier retains only a hashed version of the authentication code.

2.7.6 Verify that the initial authentication code is generated by a secure random number generator, containing at least 20 bits of entropy (typically a six digital random number is sufficient).

### 3.2.9. V2.8 One Time Verifier

2.8.1 Verify that time-based OTPs have a defined lifetime before expiring.

2.8.2 Verify that symmetric keys used to verify submitted OTPs are highly protected, such as by using a hardware security module or secure operating system based key storage.

2.8.4 Verify that time-based OTP can be used only once within the validity period.

2.8.5 Verify that if a time-based multi-factor OTP token is re-used during the validity period, it is logged and rejected with secure notifications being sent to the holder of the device.

2.8.6 Verify physical single-factor OTP generator can be revoked in case of theft or other loss. Ensure that revocation is immediately effective across logged in sessions, regardless of location.

2.8.7 Verify that biometric authenticators are limited to use only as secondary factors in conjunction with either something you have and something you know.

### 3.2.10. V2.10 Service Authentication

2.10.1 Verify that intra-service secrets do not rely on unchanging credentials such as passwords, API keys or shared accounts with privileged access.

### 3.2.11. V3.5 Token-based Session Management

3.5.1 Verify the application allows users to revoke OAuth tokens that form trust relationships with linked applications.

3.5.2 Verify the application uses session tokens rather than static API secrets and keys, except with legacy implementations.

### V3.6 Federated Re-authentication

### 3.2.12. V3.6 Federated Re-authentication

3.6.1 Verify that Relying Parties (RPs) specify the maximum authentication time to Credential Service Providers (CSPs) and that CSPs re-authenticate the user if they haven't used a session within that period.

3.6.2 Verify that Credential Service Providers (CSPs) inform Relying Parties (RPs) of the last authentication event, to allow RPs to determine if they need to re-authenticate the user.

#### 3.2.13. V3.7 Defenses Against Session Management Exploits

3.7.1 Verify the application ensures a full, valid login session or requires re-authentication or secondary verification before allowing any sensitive transactions or account modifications.

#### 3.2.14. V4.3 Other Access Control Considerations

4.3.1 Verify administrative interfaces use appropriate multi-factor authentication to prevent unauthorized use.

4.3.3 Verify the application has additional authorization (such as step up or adaptive authentication) for lower value systems, and / or segregation of duties for high value applications to enforce anti-fraud controls as per the risk of application and past fraud.

#### 3.2.15. V14.5 HTTP Request Header Validation

14.5.4 Verify that HTTP headers added by a trusted proxy or SSO devices, such as a bearer token, are authenticated by the application.

## 4. GOOGLE

### 4.1. Core app quality

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

#### 4.1.1. SC-ID1

The app provides hints to autofill account credentials and other sensitive information, such as credit card info, physical address, and phone number.

#### 4.1.2. SC-ID2

Integrate One Tap for Android for a seamless sign in experience

#### 4.1.3. SC-ID3

The app supports biometric authentication to protect financial transactions or sensitive information, such as important user documents.

### 4.2. App Security Best Practices

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

#### 4.2.1. Enforce secure communication Ask for credentials before showing sensitive information

When requesting credentials from users so that they can access sensitive information or premium content in your app, ask for either a PIN/password/pattern or a biometric credential, such as using face recognition or fingerprint recognition.

To learn more about how to request biometric credentials, see the guide about biometric authentication.

## 5. European Telecommunications Standards Institute (ETSI)

**Link:** [https://www.etsi.org/deliver/etsi\\_ts/103700\\_103799/103732/01.01.01\\_60/ts\\_103732v010101p.pdf](https://www.etsi.org/deliver/etsi_ts/103700_103799/103732/01.01.01_60/ts_103732v010101p.pdf)

### 5.1. ETSI TS 103 732

#### 5.1.1. 8.1.1 Cryptographic Support (FCS) FCS\_RNG.1 Random number generation

FCS\_RNG.1.1 The TSF shall provide a [selection: physical, non-physical true, deterministic, hybrid physical, hybrid deterministic] random number generator.

FCS\_RNG.1.2 The TSF shall provide random numbers that meet [assignment: SOG-IS Agreed Cryptographic Mechanisms [4], or, where appropriate, national or regional cryptographic standards]

#### 5.1.2. 8.1.3 Identification and Authentication (FIA) FIA\_UAU.1 Timing of authentication

FIA\_UAU.1.1 The TSF shall allow [assignment: list of TSF mediated actions not accessing user data asset stored before the action is performed unless permitted by the human user] on behalf of the human user to be performed before the human user is authenticated.

NOTE 1: Actions not listed in the assignment are not allowed before human user authentication.

EXAMPLE 1: The actions in FIA\_UAU.1.1 can include taking a picture and making an emergency call.

FIA\_UAU.1.2 The TSF shall require the human user to be successfully authenticated before allowing any other TSF-mediated actions on behalf of that human user.

#### 5.1.3. 8.1.3 Identification and Authentication (FIA) FIA\_UAU.2 User authentication before any action

FIA\_UAU.2.1 The TSF shall require each trusted peer device to be successfully authenticated by: [selection:

- both TOE and trusted peer device being logged in at the same human user account connecting to a trusted server;
- the TOE scanning a QR code generated by the trusted peer device to establish a shared secret;
- the trusted peer device scanning a QR code generated by the TOE to establish a shared secret;
- the TOE reading the NFC label embedded in the trusted peer device to establish a shared secret;
- the trusted peer device reading the NFC label embedded in the TOE to establish a shared secret;
- the human user input a PIN generated by or stored in trusted peer device in the TOE;
- the human user input a PIN generated by or stored in the TOE into trusted peer device;
- [assignment: other authentication method(s) which establish a shared secret between the TOE and trusted peer device];

before allowing any TSF-mediated actions on behalf of that trusted peer device.

NOTE 2: The shared secret in FIA\_UAU.2.1 can be pre-installed or generated in various ways, such as an earlier successful authentication attempt by one of the other authentication methods.

NOTE 3: The authentication can be required each time a peer trusted device is connecting to the TOE, or required once to build a trusted relationship after first successfully authentication.

FIA\_UAU.2.2 The TSF shall allow each trusted peer device to [assignment: list of actions] after successful authentication.

EXAMPLE 2: Actions in FIA\_UAU.2 can include screen sharing, restoring a backup, answering a (video) call to the other device, copying files by dragging, collaborative editing.

#### 5.1.4. 8.1.3 Identification and Authentication (FIA) FIA\_UAU.5 Multiple authentication mechanisms

FIA\_UAU.5.1 The TSF shall provide [selection: password, PIN, pattern] and [selection: fingerprint, iris, face, voice, vein, [assignment: other authentication mechanisms]] to support human user authentication.

FIA\_UAU.5.2 The TSF shall authenticate any human user's claimed identity according to the [assignment: rules describing how the multiple authentication mechanisms provide authentication].

EXAMPLE 3: Rules can be: "Always try biometric authentication first", "Always require both password and biometric authentication", "User chooses which mechanism to use", etc.

#### 5.1.5. 8.1.3 Identification and Authentication (FIA) FIA\_UAU.6 Re-authenticating

FIA\_UAU.6.1 The TSF shall re-authenticate the human user under the conditions:

- attempted enrolment, change or deletion of any biometric authentication factor; or attempted change of password, PIN or pattern; or
- attempted unlocking of a locked TOE; or
- [assignment: other conditions].

#### 5.1.6. 8.1.3 Identification and Authentication (FIA) FIA\_SOS.1 Verification of secrets

FIA\_SOS.1.1 The TSF shall provide a mechanism to verify that secrets meet:

- for passwords and PINs: length 4 or more.
- for patterns: consists of at least 4 and at most 9 connected points, where each point shall only be used once

NOTE 5: The human user takes care of the complexity of the PIN/Password: how easy it is to guess for others. See OE.PASSWORD\_PIN.

NOTE 6: The TOE can provide an indication of the complexity of the password/PIN to assist human user in selecting passwords/PINS that are not easy to guess.

#### 5.1.7. 8.1.3 Identification and Authentication (FIA) FIA\_UAU.7 Protected authentication feedback

FIA\_UAU.7.1 The TSF shall provide only feedback that provides at most:

- information about the length of the password or PIN; and
  - each password/PIN character for a brief moment as it is entered by the human user; and
  - the pattern for a brief time as it is entered by the human user;
- to the human user while the authentication is in progress.

#### 5.1.8. 8.1.3 Identification and Authentication (FIA) FIA\_AFL.1\_Other Authentication failure handling

FIA\_AFL.1.1/Password/PIN/Pattern The TSF shall detect when [assignment: configurable integer between 3 and 10] unsuccessful authentication attempts occur related to [selection: Password, PIN, Pattern].

FIA\_AFL.1.2/Password/PIN/Pattern When the defined number of unsuccessful authentication attempts has been met, the TSF shall [selection: reboot, progressively lengthen the time to attempt an authentication, make all user data assets unreadable, [assignment: list of other action(s) to reduce the risk of attacks such as brute-force attack]].

#### 5.1.9. 8.1.3 Identification and Authentication (FIA) FIA\_SOS.2 TSF Generation of secrets

If the assignment in FIA\_UAU.5.1 is not none, FIA\_SOS.2.1 and FIA\_SOS.2.2 apply:

FIA\_SOS.2.1 The TSF shall provide a mechanism to generate biometric templates that meet:

- A one-attempt False Acceptance Rate (FAR) that shall not exceed:
  - 1:50.000 for 2D Facial
  - 1:100.000 for 3D Facial
  - 1:50.000 for Fingerprint
  - [assignment: other value smaller or equal than 1:1000] for other method(s), and
- A one-attempt False Rejection Rate (FRR) that shall not exceed:
  - 1:15 for 2D Facial
  - 1:33 for 3D Facial
  - 1:33 for Fingerprint
  - [assignment: other value smaller or equal than 1:20] for other method(s)

FIA\_SOS.2.2 The TSF shall be able to enforce the use of TSF generated biometric templates for biometric authentication

#### 5.1.10. 8.1.4 Security Management (FMT) FMT\_SMF.1\_Authentication Specification of Management Functions

FMT\_SMF.1.1/Authentication The TSF shall be capable of performing the following management functions by the human user:

- register the initial password/PIN/pattern; and
- change the password/PIN/pattern; and
- register the initial [selection: fingerprint, iris, face, voice, vein, [assignment: other authentication mechanisms]; and
- change the [selection: fingerprint, iris, face, voice, vein, [assignment: other authentication mechanisms]]



#### 5.1.11. 8.1.3 Identification and Authentication (FIA) FIA\_AFL.1\_Password/PIN/Pattern Authentication failure handling

FIA\_AFL.1.1/Password/PIN/Pattern The TSF shall detect when [assignment: configurable integer between 3 and 10] unsuccessful authentication attempts occur related to [selection: Password, PIN, Pattern].

FIA\_AFL.1.2/Password/PIN/Pattern When the defined number of unsuccessful authentication attempts has been met, the TSF shall [selection: reboot, progressively lengthen the time to attempt an authentication, make all user data assets unreadable, [assignment: list of other action(s) to reduce the risk of attacks such as brute-force attack]].

## 6. UK National Cyber Security Centre (NCSC)

### 6.1. Application development Recommendations

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

#### 6.1.1. Android application development 1.3 Secure application development Server-side controls

Applications which store credentials should have robust server-side control procedures in place to revoke the credential, should the device or data be compromised.

#### 6.1.2. Secure iOS application development 1.3 Secure application development recommendations Server side controls

Applications which store credentials must have robust server-side control procedures in place in order to revoke credentials should the device or data be compromised.

#### 6.1.3. Secure Windows application development 1.3 Authentication

If sensitive data is handled by the application, two-factor authentication should be required when the user logs in. You should integrate Windows Hello into UWP applications to achieve this. Windows Hello provides a biometric system built in to the operating system, and utilises the device's Trusted Platform Module (TPM) chip for private key generation and storage (if available). This is a recommended option for key management as the TPM protects against several known attacks. Windows Hello can also require a PIN, which is backed by a TPM, if the organisation does not choose to use biometrics.

The Windows Hello two-factor authentication mechanism provides an alternative to smartcards. However, if Windows Hello is unavailable, smartcards can still be used to provide an additional layer of security.

If an application requires user authentication on launch, you should also implement additional checks for when the application has been backgrounded, or its use has been suspended for a length of time. This is necessary to ensure that the current user is still the authenticated user that launched the application. Where two-factor authentication has in place for the session, it would not be user-friendly to require additional two-factor authentication each time the user returns.

For authentication provided by an online identity provider, Single Sign-On (SSO) authentication should be enabled with the use of the Web Authentication Broker APIs native to UWP.

#### 6.1.4. Secure Windows application development 1.5 Server-side controls

Applications storing credentials should have robust server-side control procedures in place to revoke credentials or data stored on the device if it's compromised. If credentials are stored using Credential Locker, they can be deleted from all connected devices by using the PasswordVault.Remove functionality.

## 7. US National Institute of Standards and Technology (NIST)

### 7.1. NIST Special Publication 800-190

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

#### 7.1.1. 4.2.3 Insufficient authentication and authorization restrictions

All access to registries that contain proprietary or sensitive images should require authentication. Any write access to a registry should require authentication to ensure that only images from trusted entities can be added to it. For example, only allow developers to push images to the specific repositories they are responsible for, rather than being able to update any repository. Organizations should consider federating with existing accounts, such as their own or a cloud provider's directory service to take advantage of security controls already in place for those accounts. All write access to registries should be audited and any read actions for sensitive images should similarly be logged.

Registries also provide an opportunity to apply context-aware authorization controls to actions. For example, organizations can configure their continuous integration processes to allow images to be signed by the authorized personnel and pushed to a registry only after they have passed a vulnerability scan and compliance assessment. Organizations should integrate these automated scans into their processes to prevent the promotion and deployment of vulnerable or misconfigured images.

#### 7.1.2. 4.3.2 Unauthorized access

Access to cluster-wide administrative accounts should be tightly controlled as these accounts provide ability to affect all resources in the environment.

Organizations should use strong authentication methods, such as requiring multifactor authentication instead of just a password. Organizations should implement single sign-on to existing directory systems where applicable. Single sign-on simplifies the orchestrator authentication experience, makes it easier for users to use strong authentication credentials, and centralizes auditing of access, making anomaly detection more effective.

Traditional approaches for data at rest encryption often involve the use of host-based capabilities that may be incompatible with containers. Thus, organizations should use tools for encrypting data used with containers that allow the data to be accessed properly from containers regardless of the node they are running on. Such encryption tools should provide the same barriers to unauthorized access and tampering, using the same cryptographic approaches as those defined in NIST SP 800-111 [19].

## 8. National Information Assurance Partnership (NIAP)

### 8.1. 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](https://www.niap-ccevs.org/MMO/PP/394.R/pp_app_v1.2_table-reqs.htm)

#### 8.1.1. Secure by Default Configuration FMT\_CFG\_EXT.1.1

The application shall provide only enough functionality to set new credentials when configured with default credentials or no credentials.  
Application Note: Default credentials are credentials (e.g., passwords, keys) that are automatically (without user interaction) loaded onto the platform during application installation. Credentials that are generated during installation using requirements laid out in FCS\_RBG\_EXT.1 are not by definition default credentials.

#### 8.1.2. Password Conditioning FCS\_CKM.1(3)

The TSF shall generate salts using a RBG that meets FCS\_RBG\_EXT.1 and with entropy corresponding to the security strength selected for PBKDF in FCS\_CKM.1.1(3).

Application Note: This should be included if selected in FCS\_STO\_EXT.1 Conditioning can be performed using one of the identified hash functions or the process described in NIST SP 800-132; the method used is selected by the ST Author. SP 800-132 requires the use of a pseudo-random function (PRF) consisting of HMAC with an approved hash function. The ST author selects the hash function used, also includes the appropriate requirements for HMAC and the hash function.

Appendix A of SP 800-132 recommends setting the iteration count in order to increase the computation needed to derive a key from a password and, therefore, increase the workload of performing a password recovery attack. A significantly higher value is recommended to ensure optimal security. This value is expected to increase to a minimum of 10,000 in a future iteration based on SP800-63.