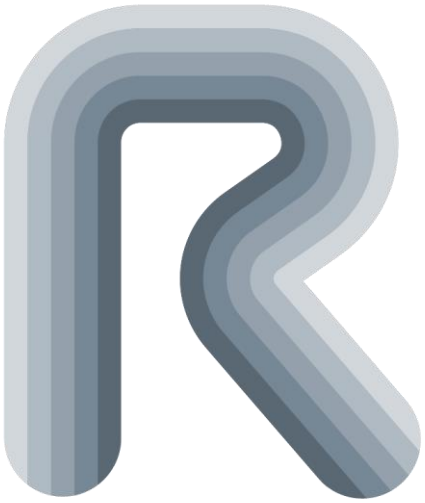
Monkton

Monkton IA Docs Reinforced by Rebar for iOS Version 1.0.0 Built on Monkton's Rebar Platform

Security Target for Monkton IA Docs for iOS

Version 1.0.11



1. Security Target Introduction

This section identifies the Target of Evaluation (TOE) along with identification of the Security Target (ST) itself. The section includes documentation organization, ST conformance claims, and ST conventions.

The Target of Evaluation (TOE) is Monkton IA Docs Reinforced by Rebar for iOS, version 1.0.0, Built on Monkton's Rebar Platform, provided by Monkton, Inc.

The Security Target contains the following sections:

- Security Target Introduction
- TOE Description
- Security Problem Definition
- Security Objectives
- IT Security Requirements
- TOE Summary Specification
- Protection Profile Claims
- Rationale

Security Target, TOE and CC Identification

Title	Description
Title	Monkton IA Docs Reinforced by Rebar for iOS, version 1.0.0, Built on Monkton's Rebar Platform Security Target
Version	1.0.11
Date	December 7, 2017
TOE Identification	Monkton IA Docs Reinforced by Rebar for iOS
TOE Version	1.0
TOE Developer	Monkton, Inc.
Evaluation Sponsor	Monkton, Inc.
CC Identification	Common Criteria for Information Technology Security Evaluation, Version 3.1, Revision 4, September 2012

1.1 Conformance Claims

The TOE is conformant to the following Common Criteria specifications

- Protection Profile for Application Software Version 1.2 (PP APP SW)
- Extended Package for Software File Encryption Version 1.0 (PP APP SWFE)
- Common Criteria for Information Technology Security Evaluation Part 2: Security functional components, Version 3.1, Revision 4, September 2012 (Part 2 Extended, Part 3 Extended)
- Common Criteria for Information Technology Security Evaluation Part 3: Security assurance components, Version 3.1 Revision 4, September 2012 (Part 3 Extended)

1.1.1 Applicable Technical Decisions

Reference	Title
0221	FMT_SMF.1.1 Assignments moved to Selections

0218	Update to FPT_AEX_EXT.1.3 Assurance Activity
0217	Compliance to RFC5759 and RFC5280 for using CRLs
0215	Update to FCS_HTTPS_EXT.1.2
0204	Protection of Selected User Data
0175	Revision of FCS_CKM_EXT.4 requirement in APP SW FE EP v1.0
0192	Update to FCS_STO_EXT.1 Application Note
0178	Integrity for installation tests in AppSW PP
0177	FCS_TLSS_EXT.1 Application Note Update
0174	Optional Ciphersuites for TLS
0172	Additional APIs added to FCS_RBG_EXT.1.1
0163	Update to FCS_TLSC_EXT.1.1 Test 5.4 and FCS_TLSS_EXT.1.1 Test
0131	Update to FCS_TLSS_EXT.1.1 Test 4.5
0123	GCM Mode Added to FCS_KYC_EXT.1.1, FCS_COP.1.1(1), FPT_KYP_EXT.1.1
0121	FMT_MEC_EXT.1.1 Configuration Options
0119	FCS_STO_EXT.1.1 in PP_APP_v1.2
0107	FCS_CKM - ANSI X9.31-1998, Section 4.1.for Cryptographic Key Generation
0092	FCS_KYC_EXT.1 - Key Integrity
0076	Correction to SWFE Keychain Requirement
0069	Revision to FCS_COP.1(1) AA in SWFE EP v1.0
0067	Revision to FCS_CKM.1(A) SFR & AA in SWFE EP v1.0
0065	Revision of FDP_PRT_EXT.1.2 requirement in APP SWFE EP v1.0

1.2 Conventions

The following conventions are applied for the Monkton IA Docs Reinforced by Rebar Security Target document

Security Functional Requirements – Part 2 of the CC defines the approved set of operations that may be applied to functional requirements: iteration, assignment, selection, and refinement

- *Iteration*: allows a component to be used more than once with varying operations. In the ST, iteration is indicated by a number in parentheses placed at the end of the component. For example, FDP_ACC.1(1) and FDP_ACC.1(2) indicate that the ST includes two iterations of the FDP_ACC.1 requirement, (1) and (2).
- *Assignment*: allows the specification of an identified parameter. Assignments are indicated using bold and are surrounded by brackets (e.g., [**assignment**]). Note that an assignment within a selection would be identified in italics and with embedded bold brackets (e.g., [[**selected- assignment**]]).
- *Selection*: allows the specification of one or more elements from a list. Selections are indicated using bold italics and are surrounded by brackets (e.g., [**selection**]).
- *Refinement*: allows the addition of details. Refinements are indicated using bold, for additions, and strike-through, for deletions (e.g., "... **all** objects ..." or "... ~~some~~ **big** things ..."). Note that 'cases' that are not applicable in a given SFR have simply been removed without any explicit identification.

The PP APP SW and PP APP SWFE use an additional convention – the `case` – which defines parts of an SFR that apply only when corresponding selections are made or some other identified conditions exist. Only the applicable cases are identified in this ST and they are identified using bold text.

Other sections of the ST – Other sections of the ST use bolding to highlight text of special interest, such as captions and relevant information.

1.3 Terminology

PP APP SW and PP APP SWFE provide definitions for terms specific to the application software technology as well as general Common Criteria terms. The technology-specific terms are:

PP APP SW and PP APP SWFE Terms
Address Space Layout Randomization
Application
Administrator
Application Programming Interface
Authorization factor
Authorized User
Credential
Data Encryption
Data Execution Prevention
Developer
Entropy Source
File/Set of files
File Authentication Key
File Encryption Key
Key Encryption Key
Keying material
Mobile Code
Noise Source
Operational Environment
Operating System
Password
Passphrase
Personally Identifiable Information
Platform
Random Bit Generator
Sensitive Data
System files
Temporary File
Trusted Host
Unauthorized User
User Data
Vendor
Volatile memory
Zeroize

Common Criteria Terms
Common Criteria
Common Evaluation Methodology

Protection Profile
Security Target
Target of Evaluation
TOE Security Functionality
TOE Summary Specification
Security Functional Requirement
Security Assurance Requirement

1.4 Abbreviations

This section identifies abbreviations and acronyms used in this ST

Term	Explanation
API	Application Programming Interface
App	Software Application
ASLR	Address Space Layout Randomization
CC	Common Criteria
CEM	Common Evaluation Methodology
DEP	Data Execution Prevention
FEK	File Encryption Key
DOD	Department of Defense
KEK	Key Encryption Key
MDM	Mobile Device Management
OS	Operating System
PII	Personally Identifiable Information
PP	Protection Profile
PP APP SW	Protection Profile for Application Software
PP APP SWFE	Extended Package for Software File Encryption
RMD	Rebar Management Database
RKM	Rebar Key Management
RTX	Rebar Authentication Token Context
SAR	Software Assurance Requirement
SFR	Software Functional Requirement
ST	Security Target
TOE	Target of Evaluation
TSF	TOE Security Functionality
TSS	TOE Summary Specification

2. TOE Description

The Target of Evaluation (TOE) for this assessment is Monkton IA Docs Reinforced by Rebar for iOS, version 1.0.0, Built on Monkton's Rebar Platform.

2.1 Product Overview

Monkton IA Docs is a mobile app that is installed on the Apple iPhone and the Apple iPad. This app enables end users to securely download and view documents stored in a cloud/service provider.

Monkton IA Docs is Managed Configuration Enabled with MDM providers, allowing MDM providers to push configuration settings down to devices. This enables enterprises to manage apps on managed devices in a secure manner.

Monkton IA Docs is built on Monkton's Rebar Platform that provides apps to be compliant with NIAP. The Rebar Middleware ("Middleware") is a server component, ***not under evaluation***, but is part of the OE, that is installed within the enterprise's data center or cloud provider. The Middleware enables authentication, authorization, and facilitates access to web services that the mobile app may need.

In the context of Monkton IA Docs, the Middleware provides hooks to cloud document providers that enables access to cloud stored documents. By design with apps built on Rebar and the Middleware, this architecture provides higher levels of insurance to protecting information stored in Monkton IA Docs. Due to the architecture of relying on the Middleware to request files on behalf of Monkton IA Docs, access to cloud providers can be done in a "server to server" manner, removing the app (Monkton IA Docs) from holding access keys to access the cloud providers directly.

Files stored locally within Monkton IA Docs are protected by the OS and the cryptography provided as part of the PP APP SWFE conformance. All files are stored locally encrypted with AES256 encryption.

2.2 TOE Overview

The TOE is the Monkton IA Docs Reinforced by Rebar for iOS, version 1.0.0 mobile app. The TOE interacts with the Rebar Middleware ("Middleware"), ***a server component not under evaluation*** to serve up documents from a cloud storage provider, also known as a Service Provider (SP). The TOE is a composition of the application code (IA Docs) and the Rebar SDK (Also referred to as just "Rebar"), which provides the implementation of the included NIAP Protection Profiles in this Security Target. When "Rebar" is referenced in this document it references the TOE, it can be used in an interchanged method in referencing the TOE itself.

The TOE interaction with the "Middleware" leverages the "Middleware Identity Provider" ("IdP"), part of the Middleware Authentication and Authorization process. When a user initially authenticates the TOE with the Middleware via username / password, the app is granted a Rebar Token Context (RTX) for that user using the app on that specific device.

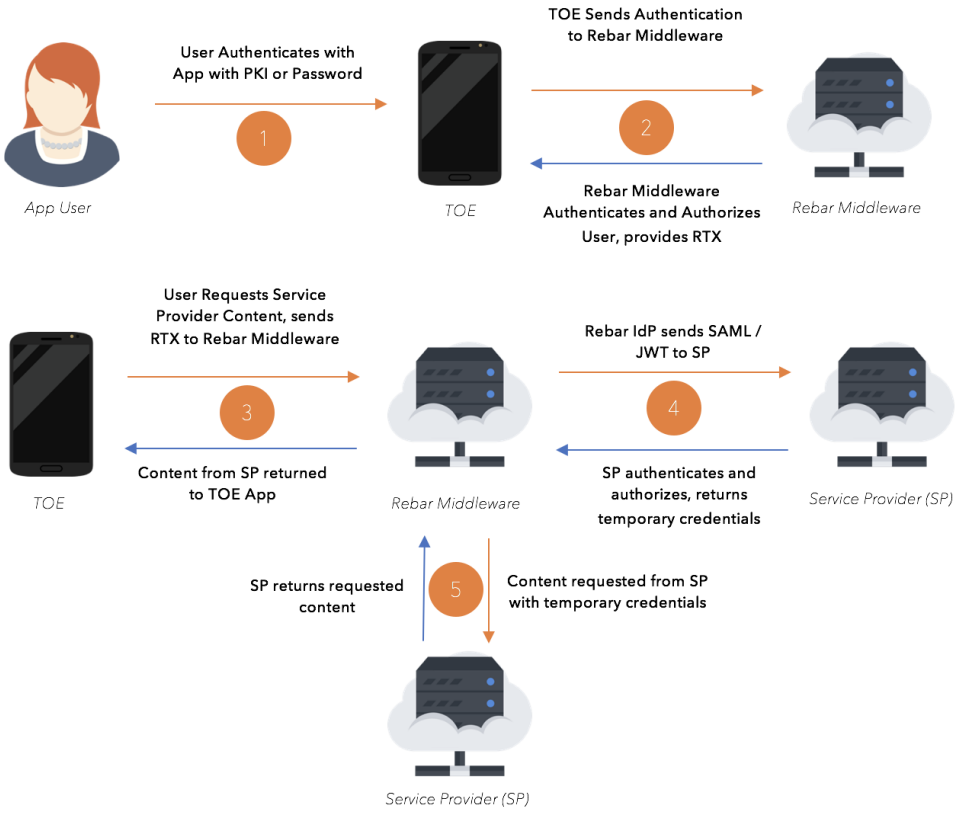
When the TOE wants to request a resource from a SP, the app invokes a web service method via the Rebar SDK. The Rebar SDK, leveraging the RTX, calls the Middleware (3). The Middleware authenticates and authorizes the user based on their RTX. If the RTX is valid and the user is allowed to access the resource, they will begin to authenticate with the Rebar IdP service.

The Middleware will begin the IdP process for the requested service provider. If the user is tied to an active directory account, the user's UPN value is leveraged to authenticate the user to the service provider (4). The

Rebar IdP creates the SAML Response or JWT Assertion to present to the service provider. The Rebar IdP will send the request to the service provider endpoint for programmatic authentication.

If the IdP SAML Response or JWT Assertion is authenticated and authorized by the service provider, the provider will issue temporary tokens or credentials to perform operations against the service provider. Rebar will temporarily cache the credentials for the request avoiding multiple authentications for the individual user. Rebar will then return the resource with the temporary credentials from the service provider (5).

TOE_OVERVIEW

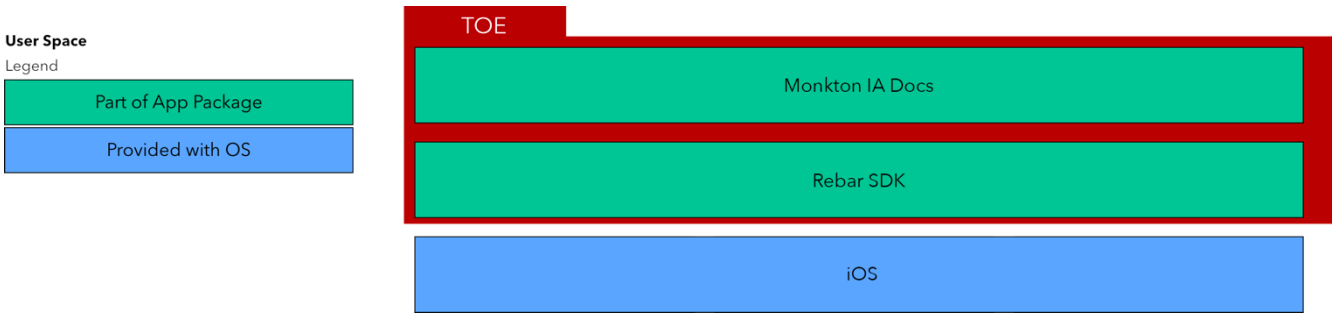


2.3 TOE Architecture

The section describes the TOE architecture including physical and logical boundaries. Figure TOE_ARCHITECTURE shows the relationship of the TOE to its operational environment along with the TOE boundary. The security functional requirements identify the libraries included in the application package. Part of the TOE is the Rebar SDK. The Rebar SDK provides compliance to the necessary NIAP protection profile SFRs being evaluated in this document. Rebar, Rebar SDK, are analogous with the TOE in terms of being referenced in this document. Any references to Rebar or the Rebar SDK should be considered the TOE.

Monkton IA Docs is a combination of the Monkton provided Rebar SDK. The IA Docs executable and Rebar SDK are specific parts of vendor provided elements of the TOE. The TOE executes on top of iOS and interacts with platform provides APIs to perform specific tasks that are provided by the platform.

TOE_ARCHITECTURE



2.3.1 Physical Boundaries

The TOE consists of the Monkton IA Docs app and the configuration settings defined in the Monkton IA Docs for iOS Version 1.0.0 User Guide, Version 1.0. ***The Middleware and any functions not specified in this security target are outside the scope of this TOE***.

2.3.2 Software Requirements

The TOE runs on iOS 10.3.2.

2.3.3 Hardware Requirements

The TOE was evaluated on Apple's iPhone 7 (A10 Fusion with 64-bit architecture) and iPad Pro 10" (A10X Fusion with 64-bit architecture).

2.3.4 Logical Boundaries

This section summarizes the security functions provided by the TOE

2.3.4.1 Cryptographic Support

The TOE provides several functions for cryptographic support. The TOE, by virtue of being built on Rebar, implements DAR and DIT as a functional component. When HTTPS network connections are created, they are made over TLS 1.2 connections with the requisite cipher suites. The TOE uses OpenSSL 1.0.2L to provide its cryptographic support.

The TOE, through Rebar, provides all requisite cryptographic functions for hashing, signing, HMAC, random number generation, and symmetric encryption.

To protect the keys and data generated by the TOE, the TOE will aggressively and securely delete key data and files written to non-volatile memory. This leverages both platform implemented functions as well as functions integrated into Rebar.

The relevant CAVP certificates are listed below for the TOE and the platform provided DRBG.

	Algorithm	Length	SFR	CAVP
AES	CBC, GCM	128, 192, 256	FCS_COP.1(1) FCS_COP.1(5)	4751
HMAC	HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512	Key: 160, 256, 512; Digest: 160, 256, 512	FCS_COP.1.1(4)	3165
ECDSA	PKG, PKV	P-256 P-384 P-521	FCS_COP.1(3) FCS_CKM.1(1)	1245

	SigGen, SigVer	P-256 P-384 P-521 with SHA-256, SHA-384, SHA-512	FCS_CKM.2.1 FCS_COP.1(3) FCS_CKM.1(1)	1245
RSA	ANSIX9.31: SigGen, SigVer	2048, 3072 with SHA-256, SHA-384, SHA-512	FCS_COP.1(3) FCS_CKM.1(1)	2595
	RSASSA-PKCS1_V1_5: SigGen, SigVer	2048, 3072 with SHA-256, SHA-384, SHA-512	FCS_COP.1(3) FCS_CKM.1(1)	2595
	RSASSA-PSS: SigGen, SigVer	2048, 3072 with SHA-256, SHA-384, SHA-512	FCS_COP.1(3) FCS_CKM.1(1)	2595
SHS	SHA	SHA-1, SHA-256, SHA-384, SHA-512	FCS_COP.1.1(2)	3008
CVL/KAS /ECC	ECC/KPG/ EphemUnified	P-256, P-384, P-521	FCS_COP.1(3) FCS_CKM.1(1)	1388
DRBG	AES-256 CTR DRBG	256	FCS_RBG_EXT.1.1, FCS_CKM.1.1(2), FCS_RBG_EXT.2.1, FCS_RBG_EXT.2.2, FCS_CKM_EXT.2.1	1632

2.3.4.2 User Data Protection

The TOE requests no hardware or software resources during the use of the application. The TOE requires network access but this is not a request that is prompted to the user.

2.3.4.3 Identification and Authorization

The TOE, through Rebar, implements X509 certificate validation for all server certificates presented for TLS 1.2 connections. Additionally, Rebar implements SSL Pinning, validating certificates based on SHA512 hashes of the certificates.

For user authorization, the TOE leverages PBKDF2 with HMACSHA256 to validate user credentials based on a passcode. The conditioned key is used as the FEK for the RMD. The passcode can be configured by the administrator for complexity requirements.

2.3.4.4 Security Management

The TOE is, by default, configured to be secure whenever it is freshly installed on a device. The TOE, through Rebar, provides configuration settings available through the Managed App Configuration settings. Rebar implements a secure version of NSUserDefaults that ensures settings are stored in an AES256 encrypted database.

2.3.4.5 Protection of the TSF

The TOE leverages only approved iOS APIs and available libraries. The TOE includes several third-party libraries that provide specific functionality for the TOE. Each of these libraries leverage only approved iOS APIs.

The TOE leverages the iOS update manager (App Store) or enterprise distribution mechanisms (MDM/Enterprise App Store) to update and install approved apps.

All key material used within the TOE is protected and destroyed as part of the cryptographic support. The password conditioned FEK is never stored in non-volatile memory.

2.3.4.6 Trust Paths

All data in transit for the TOE is sent with TLS 1.2.

2.3.4.7 Privacy

The TOE does not transmit PII.

2.4 TOE Documentation

The TOE includes the following documentation

- *Rebar Platform Administrative Guide, Version 1.0*
- *Monkton IA Docs for iOS Version 1.0.0 User Guide, Version 1.0*

3. Security Problem Definition

This security target includes by reference the Security Problem Definition from the PP APP SW and PP APP SWFE. The Security Problem Definition consists of threats that a conformant TOE is expected to address and assumptions about the operational environment of the TOE.

In general, the PP APP SW and PP APP SWFE has presented a Security Problem Definition appropriate for application software which runs on mobile devices. Monkton IA Docs is an iOS App running on a mobile device. As such, the PP APP SW and PP APP SWFE Security Problem Definitions apply to the TOE.

4 Security Objectives

This security target includes by reference the Security Objectives from the PP APP SW and PP APP SWFE. The PP APP SW and PP APP SWFE Security Objectives for the operational environment are reproduced below. Since these objectives characterize technical and procedural measures each consumer must implement in their operational environment.

In general, the PP APP SW and PP APP SWFE has presented a Security Objectives statement appropriate for appropriate for application software which runs on mobile devices. Consequently, the PP APP SW and PP APP SWFE security objectives are suitable for the TOE.

4.1 Security Objectives for the TOE

Objective	Objective Description
O.INTEGRITY	Conformant TOEs ensure the integrity of their installation and update packages, and also leverage execution environment-based mitigations. Software is seldom if ever shipped without errors, and the ability to deploy patches and updates to fielded software with integrity is critical to enterprise network security. Processor manufacturers, compiler developers, execution environment vendors, and operating system vendors have developed execution environment-based mitigations that increase the cost to attackers by adding complexity to the task of compromising systems. Application software can often take advantage of these mechanisms by using APIs provided by the runtime environment or by enabling the mechanism through compiler or linker options.
O.QUALITY	To ensure quality of implementation, conformant TOEs leverage services and APIs provided by the runtime environment rather than implementing their own versions of these services and APIs. This is especially important for cryptographic services and other complex operations such as file and media parsing. Leveraging this platform behavior relies upon using only documented and supported APIs.
O.MANAGEMENT	To facilitate management by users and the enterprise, conformant TOEs provide consistent and supported interfaces for their security-relevant configuration and maintenance. This includes the deployment of applications and application updates through the use of platform-supported deployment mechanisms and formats, as well as providing mechanisms for configuration. This also includes providing control to the user regarding disclosure of any PII.
O.PROTECTED_STORAGE	To address the issue of loss of confidentiality of user data in the event of loss of physical control of the storage medium, conformant TOEs will use data-at-rest protection. This involves encrypting data and keys stored by the TOE in order to prevent unauthorized access to this data. This also includes unnecessary

	network communications whose consequence may be the loss of data.
O. PROTECTED_COMMS	To address both passive (eavesdropping) and active (packet modification) network attack threats, conformant TOEs will use a trusted channel for sensitive data. Sensitive data includes cryptographic keys, passwords, and any other data specific to the application that should not be exposed outside of the application.
O.AUTHORIZATION	The TOE must enforce the entry of authorization factor(s) by authorized users to be able to encrypt and decrypt user data.
O.CORRECT_TSF_OPERATION	The TOE will provide the capability to test the TSF to ensure the correct operation of the TSF in its operational environment.
O.PROTECT_DATA	The TOE will decrypt/encrypt all user data that is provided to the file encryption program in order to protect it while it is not being activity accessed by the user
O.FEK_SECURITY	The TOE will encrypt the FEK using a KEK created from one or more authorization factors so that a threat agent who does not have the authorization factor(s) will be unable to gain access to the user data by obtaining the FEK. The size of the FEK will be large enough to make a brute force attack infeasible.
O.KEY_MATERIAL_PROTECTION	The TOE shall ensure that unencrypted keys or keying material are properly removed from memory after use.
O.MANAGE	The TOE will provide all the functions and facilities necessary to support the authorized administrators in their management of the security of the TOE, and restrict these functions and facilities from unauthorized use.
O. SAFE_AUTHFACTOR_VERIFICATION	The TOE shall perform verification of the authorization factors in such a way that the KEK, FEK, or user data are not inadvertently exposed.
O.WIPE_MEMORY	The TOE shall ensure that non-volatile memory space corresponding to sensitive plaintext material (encryption input) is wiped from the TOE's memory. This includes temporary files that may have been created.

4.2 Security Objectives for the Operational Environment

Objective	Objective Description
OE.PLATFORM	The TOE relies upon a trustworthy computing platform for its execution. This includes the underlying operating system and any discrete execution environment provided to the TOE.
OE.PROPER_USER	The user of the application software is not willfully negligent or hostile, and uses the software within compliance of the applied enterprise security policy.
OE.PROPER_ADMIN	The administrator of the application software is not careless, willfully negligent or hostile, and administers the

	software within compliance of the applied enterprise security policy.
OE.AUTHORIZATION_FACTOR_STRENGTH	An authorized user will be responsible for ensuring that all externally derived authorization factors have sufficient strength and entropy to reflect the sensitivity of the data being protected. This can apply to password or passphrase-based, ECC CDH, and RSA authorization factors.
OE.POWER_SAVE	The non-mobile operational environment must be configurable so that there exists at least one mechanism that will cause the system to power down after a period of time in the same fashion as the user electing to shutdown the system (A.SHUTDOWN). Any such mechanism (e.g., sleep, hibernate) that does not conform to this requirement must be capable of being disabled. The mobile operational environment must be configurable such that there exists at least one mechanism that will cause the system to lock upon a period of time.
OE.STRONG_ENVIRONMENT_ CRYPTO	The Operating environment will provide a cryptographic function capability that is commensurate with the requirements and capabilities of the TOE.
OE.TRAINED_USERS	Authorized users of the host machine will be trained to follow all provided guidance.

5. IT Security Requirements

This section defines the Security Functional Requirements (SFRs) and Security Assurance Requirements (SARs) that serve to represent the security functional claims for the Target of Evaluation (TOE) and to scope the evaluation effort.

Protection Profile for Application Software (PP APP SW) and Extended Package for Software File Encryption (PP APP SWFE) provide the baseline security functional requirements for this section. All completed requirements are contained herein. The requirements are broken into two separate sections, one for the PP APP SW and PP APP SWFE.

The security assurance requirements are the set of SARs specified in PP APP SW and PP APP SWFE.

5.1 Extended Requirements

All of the extended requirements in this ST have been drawn from the PP APP SW and PP APP SWFE. The PP APP SW and PP APP SWFE define the following extended SFRs. Since these SFRs are not redefined in this ST, readers should consult PP APP SW and PP APP SWFE for more information in regard to these CC extensions.

- FIA_AUT_EXT.1 - User Authorization
- FIA_FCT_EXT.1(2) - Extended: User Authorization with Password/Passphrase Authorization Factors
- FIA_X509_EXT.1 - X.509 Certificate Validation
- FIA_X509_EXT.2 - X.509 Certificate Authentication
- FCS_CKM_EXT.1 - Cryptographic Key Generation Services (PP APP SW)
- FCS_CKM_EXT.1 - Key Encrypting Key (KEK) Support (PP APP SWFE)

- FCS_CKM_EXT.1(A) Extended: Cryptographic Key Generation (Password/Passphrase Conditioning)
- FCS_CKM_EXT.2 - Cryptographic key generation (FEK)
- FCS_CKM_EXT.4 - Extended: Cryptographic Key Destruction
- FCS_HTTPS_EXT.1 - HTTPS Protocol
- FCS_IV_EXT.1 - Extended: Initialization Vector Generation
- FCS_KYC_EXT.1 - Key Chaining and Key Storage
- FCS_RBG_EXT.1 - Random Bit Generation Services
- FCS_RBG_EXT.2 - Random Bit Generation from Application
- FCS_STO_EXT.1 - Storage of Secrets
- FCS_TLSC_EXT.1 - TLS Client Protocol
- FCS_TLSC_EXT.3 - TLS Client Protocol
- FCS_TLSC_EXT.4 - TLS Client Protocol
- FDP_DAR_EXT.1 - Encryption of Sensitive Application Data
- FDP_DEC_EXT.1 - Access to Platform Resources
- FDP_NET_EXT.1 - Network Communications
- FDP_PRT_EXT.1 - Extended: Protection of Selected User Data
- FMT_CFG_EXT.1 - Secure by Default Configuration
- FMT_MEC_EXT.1 - Supported Configuration Mechanism
- FPR_ANO_EXT.1 - User Consent for Transmission of Personally Identifiable Information
- FPT_AEX_EXT.1 - Anti-Exploitation Capabilities
- FPT_API_EXT.1 - Use of Supported Services and APIs
- FPT_FEK_EXT.1 - File Encryption Key (FEK) Support
- FPT_LIB_EXT.1 - Use of Third Party Libraries
- FPT_KYP_EXT.1 - Extended: Protection of Key and Key Material (FPT_KYP_EXT)
- FPT_TUD_EXT.1 - Integrity for Installation and Update
- FTP_DIT_EXT.1 - Protection of Data in Transit

5.2 TOE Security Functional Requirements

The following table identifies the SFRs that are satisfied by the TOE for the PP APP SW and PP APP SWFE.

Due to inconsistencies in identifiers for the SFR's in *FCS: Cryptographic Support* in PP APP SW and PP APP SWFE, specifically the *FCS_CKM_EXT.1* requirement being two different requirements, but having the same identifiers, they are noted as PP APP SW and PP APP SWFE.

Requirement Class	**Requirement Component**
FCS: Cryptographic Support	FCS_CKM.1(1) - Cryptographic Asymmetric Key Generation
	FCS_CKM.1(2) - Cryptographic Symmetric Key Generation
	FCS_CKM.2 - Cryptographic Key Establishment
	FCS_CKM_EXT.1 - Cryptographic Key Generation Services (PP APP SW)
	FCS_CKM_EXT.1 - Key Encrypting Key (KEK) Support (PP APP SWFE)
	FCS_CKM_EXT.1(A) Extended: Cryptographic Key Generation (Password/Passphrase Conditioning)
	FCS_CKM_EXT.2 - Cryptographic key generation (FEK)
	FCS_CKM_EXT.4 - Extended: Cryptographic Key Destruction
	FCS_COP.1(1) - Cryptographic Operation (Data Encryption)

	FCS_COP.1(2) - Cryptographic Operation - Hashing
	FCS_COP.1(3) - Cryptographic Operation – Signing
	FCS_COP.1(4) - Cryptographic Operation (Keyed-Hash Message Authentication)
	FCS_COP.1(5) - Cryptographic operation (Key Wrapping)
	FCS_HTTPS_EXT.1 - HTTPS Protocol
	FCS_IV_EXT.1 - Extended: Initialization Vector Generation
	FCS_KYC_EXT.1 - Key Chaining and Key Storage
	FCS_RBG_EXT.1 - Random Bit Generation Services
	FCS_RBG_EXT.2 - Random Bit Generation from Application
	FCS_STO_EXT.1 - Storage of Secrets
	FCS_TLSC_EXT.1 - TLS Client Protocol
	FCS_TLSC_EXT.3 - TLS Client Protocol
	FCS_TLSC_EXT.4 - TLS Client Protocol
FDP: User Data Protection	FDP_DAR_EXT.1 - Encryption of Sensitive Application Data
	FDP_DEC_EXT.1 - Access to Platform Resources
	FDP_NET_EXT.1 - Network Communications
	FDP_PRT_EXT.1 - Extended: Protection of Selected User Data
FMT: Security Management	FMT_CFG_EXT.1 - Secure by Default Configuration
	FMT_MEC_EXT.1 - Supported Configuration Mechanism
	FMT_SMF.1 - Specification of Management Functions
FPT: Protection of the TSF	FPT_AEX_EXT.1 - Anti-Exploitation Capabilities
	FPT_API_EXT.1 - Use of Supported Services and APIs
	FPT_FEK_EXT.1 - File Encryption Key (FEK) Support
	FPT_KYP_EXT.1 - Extended: Protection of Key and Key Material (FPT_KYP_EXT)
	FPT_LIB_EXT.1 - Use of Third Party Libraries
	FPT_TUD_EXT.1 - Integrity for Installation and Update
FTP: Trusted Path/Channel	FTP_DIT_EXT.1 - Protection of Data in Transit
Privacy	FPR_ANO_EXT.1 - User Consent for Transmission of Personally Identifiable Information
FIA: Identification and Authentication	FIA_AUT_EXT.1 - User Authorization
	FIA_FCT_EXT.1(2) - Extended: User Authorization with Password/Passphrase Authorization Factors
	FIA_X509_EXT.1 - X.509 Certificate Validation
	FIA_X509_EXT.2 - X.509 Certificate Authentication

5.2.1 FCS: Cryptographic Support

5.2.1.1 FCS_CKM.1(1) - Cryptographic Asymmetric Key Generation

FCS_CKM.1.1(1) The application shall generate asymmetric cryptographic keys in accordance with a specified cryptographic key generation algorithm [*[ECC schemes] using ["NIST curves" P-256, P-384 and [P-521]] that meet the following: [FIPS PUB 186-4, "Digital Signature Standard (DSS)", Appendix B.4]*].

5.2.1.2 FCS_CKM.1(2) - Cryptographic Symmetric Key Generation

FCS_CKM.1.1(2)	The application shall generate symmetric cryptographic keys using a Random Bit Generator as specified in FCS_RBG_EXT.1 and specified cryptographic key sizes [*256 bit*].

5.2.1.3 FCS_CKM.2 - Cryptographic Key Establishment

FCS_CKM.2.1	The application shall [*implement functionality*] to perform cryptographic key establishment in accordance with a specified cryptographic key establishment method: [*RSA-based key establishment schemes*] that meets the following: [*NIST Special Publication 800-56B, "Recommendation for Pair-Wise Key Establishment Schemes Using Integer Factorization Cryptography"*] and [*[Elliptic curve-based key establishment schemes] that meets the following: [NIST Special Publication 800-56A, Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography]*].

5.2.1.4 FCS_CKM_EXT.1 - Cryptographic Key Generation Services (PP APP SW)

FCS_CKM_EXT.1.1	The application shall [*implement asymmetric key generation*].

5.2.1.5 FCS_CKM_EXT.1 - Key Encrypting Key (KEK) Support (PP APP SWFE)

FCS_CKM_EXT.1.1	The TSF shall support KEK in the following manner based on the selection chosen in FPT_FEK_EXT.1: [*derive a KEK using a password-based authorization factor conditioned as defined in FCS_CKM.1(A) and in accordance with FIA_FCT_EXT.1(2)*]
FCS_CKM_EXT.1.2	All KEKs shall be [*256-bit*] keys corresponding to at least the security strength of the keys encrypted by the KEK.

5.2.1.6 FCS_CKM_EXT.1(A) Extended: Cryptographic Key Generation (Password/Passphrase Conditioning)

FCS_CKM_EXT.1.1 (A)	The TSF shall support a password/passphrase of up to [*255*] characters used to generate a password authorization factor
FCS_CKM_EXT.1.2(A)	The TSF shall allow passwords to be composed of any combination of upper case characters, lower case characters, numbers, and the following special characters: "!", "@", "#", "$", "%", "^", "&", "*", "(", and ")", and [*any other characters available on the system keyboard*].
FCS_CKM_EXT.1.3(A)	The TSF shall perform Password-based Key Derivation Functions in accordance with a specified cryptographic algorithm [*HMAC-[SHA-256]*], with [*4096*] iterations, and output cryptographic key sizes [*256*] that meet the following: [*NIST SP 800-132*]
FCS_CKM_EXT.1.4(A)	The TSF shall not accept passwords less than [*a value settable by the administrator, [6]*] and greater than the maximum password length defined in FCS_CKM_EXT.1.1(A).

FCS_CKM_EXT.1.5(A)	The TSF shall generate all salts using a RBG that meets FCS_RBG_EXT.1 (from the AS PP) and with entropy corresponding to the security strength selected for PBKDF in FCS_CKM_EXT.1.3(A).

5.2.1.7 FCS_CKM_EXT.2 - Cryptographic key generation (FEK)

FCS_CKM_EXT.2.1	The TSF shall generate FEK cryptographic keys [*using a Random Bit Generator as specified in FCS_RBG_EXT.1 and with entropy corresponding to the security strength of AES key sizes of [256 bits]; conditioned from a password/passphrase as defined in FCS_CKM.1(A)*].
FCS_CKM_EXT.2.2	The TSF shall create a unique FEK for each file (or set of files) using the mechanism on the client as specified in FCS_CKM_EXT.2.1.
FCS_CKM_EXT.2.3	The FEKs must be generated by the TOE.

5.2.1.8 FCS_CKM_EXT.4 - Extended: Cryptographic Key Destruction

FCS_CKM_EXT.4.1	The TSF shall destroy cryptographic keys in accordance with a specified cryptographic key destruction method [• ***For volatile memory, the destruction shall be executed by a [single overwrite consisting of [zeroes]]*** • ***For non-volatile memory [that consists of the invocation of an interface provided by the underlying platform that [logically addresses the storage location of the key and performs a [single overwrite consisting of [pseudo-random pattern]]]]***] that meets the following: No Standard.

5.2.1.9 FCS_COP.1(1) - Cryptographic operation (Data Encryption)

FCS_COP.1.1(1)	Refinement: The application shall [*implement AES encryption*] to perform data encryption and decryption in accordance with a specified cryptographic algorithm AES used in [*CBC (as defined in NIST SP 800-38A); GCM (as defined in NIST SP 800-38D);*] mode and cryptographic key sizes [*128 bits, 256 bits*].

5.2.1.10 FCS_COP.1(2) - Cryptographic Operation – Hashing

FCS_COP.1.1(2)	The application shall perform cryptographic hashing services in accordance with a specified cryptographic algorithm [*SHA-1, SHA-256, SHA-384, SHA-512*] and message digest sizes [*160, 256, 384, 512*] bits that meet the following: FIPS Pub 180-4.

5.2.1.11 FCS_COP.1(3) - Cryptographic Operation – Signing

FCS_COP.1.1(3)	The application shall perform cryptographic signature services (generation and verification) in accordance with a specified cryptographic algorithm [RSA schemes using cryptographic key sizes of 2048-bit or greater that meet the following: FIPS PUB 186-4, "Digital Signature Standard (DSS)", Section 4, ECDSA schemes using "NIST curves" P-256, P-384 and [*P-521*] that meet the following: FIPS PUB 186-4, "Digital Signature Standard (DSS)", Section 5].

5.2.1.12 FCS_COP.1(4) - Cryptographic Operation (Keyed-Hash Message Authentication)

FCS_COP.1.1(4)	Refinement: The application shall [*implement functionality*] to perform keyed-hash message authentication in accordance with a specified cryptographic algorithm HMAC- [*SHA-256, SHA-512*], key size [*256, 512*], and message digest size of [*256, 512*] bits that meet the following: FIPS PUB 198-1, "The Keyed-Hash Message Authentication Code", and FIPS PUB 180-4, "Secure Hash Standard"

5.2.1.13 FCS_COP.1(5) - Cryptographic operation (Key Wrapping)

FCS_COP.1.1(5)	The application shall [implement functionality to perform Key Wrapping] in accordance with a specified cryptographic algorithm [*AES Key Wrap*] and the cryptographic key size [*256 bits (AES)*] that meet the following: [*"NIST SP 800-38F" for Key Wrap (section 6.2) and Key Wrap with Padding (section 6.3)*]

5.2.1.14 FCS_HTTPS_EXT.1 - HTTPS Protocol

FCS_HTTPS_EXT.1.1	The application shall implement the HTTPS protocol that complies with RFC 2818.
FCS_HTTPS_EXT.1.2	The application shall implement HTTPS using TLS in accordance with [*FCS_TLSC_EXT.1*]
FCS_HTTPS_EXT.1.3	The application shall notify the user and [*not establish the connection*] if the peer certificate is deemed invalid.

5.2.1.15 FCS_IV_EXT.1 - Extended: Initialization Vector Generation

FCS_IV_EXT.1.1	The application shall [*generate IVs*] in accordance with Appendix H: Initialization Vector Requirements for NIST-Approved Cipher Modes.

5.2.1.16 FCS_KYC_EXT.1 - Key Chaining and Key Storage

FCS_KYC_EXT.1.1	The TSF shall maintain a primary key chain of: [*KEKs originating from one or more authorization factors(s) to the FEK(s) using the following method(s): [implement key wrapping as specified in FCS_COP.1(5)] while maintaining an overall effective strength of [[256 bits] for symmetric keys] commensurate with the strength of the FEK*] and [*no supplemental key chains*].

5.2.1.17 FCS_RBG_EXT.1 - Random Bit Generation Services

FCS_RBG_EXT.1.1 The application shall [*invoke platform-provided DRBG functionality*, *implement DRBG functionality*] for its cryptographic operations.

5.2.1.18 FCS_RBG_EXT.2 - Random Bit Generation from Application

FCS_RBG_EXT.2.1 The application shall perform all deterministic random bit generation (DRBG) services in accordance with NIST Special Publication 800-90A using [*CTR_DRBG (AES)*].

FCS_RBG_EXT.2.2 The deterministic RBG shall be seeded by an entropy source that accumulates entropy from a platform-based DRBG and [*no other noise source*] with a minimum of [*256 bits*] of entropy at least equal to the greatest security strength (according to NIST SP 800-57) of the keys and hashes that it will generate.

5.2.1.19 FCS_STO_EXT.1 - Storage of Secrets

FCS_STO_EXT.1.1 The application shall [*implement functionality to securely store [RTX and file encryption keys]*] to non-volatile memory.

5.2.1.20 FCS_TLSC_EXT.1 - TLS Client Protocol

FCS_TLSC_EXT.1.1 The application shall [*implement TLS 1.2 RFC 5246*] supporting the following cipher suites: Mandatory Cipher Suites: TLS_RSA_WITH_AES_128_CBC_SHA as defined in RFC 5246 Optional Cipher Suites: [

- ***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_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_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***
- ***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***

].

FCS_TLSC_EXT.1.2 The application shall verify that the presented identifier matches the reference identifier according to RFC 6125.

FCS_TLSC_EXT.1.3 The application shall establish a trusted channel only if the peer certificate is valid.

5.2.1.21 FCS_TLSC_EXT.3 - TLS Client Protocol

FCS_TLSC_EXT.3.1	The application shall present the signature_algorithms extension in the Client Hello with the supported_signature_algorithms value containing the following hash algorithms: [*SHA256, SHA384, SHA512*] and no other hash algorithms.

5.2.1.22 FCS_TLSC_EXT.4 - TLS Client Protocol

FCS_TLSC_EXT.4.1	The application shall present the supported Elliptic Curves Extension in the Client Hello with the following NIST curves: [*secp256r1, secp384r1, secp521r1*] and no other curves.

5.2.2 FDP: User Data Protection

5.2.2.1 FDP_DAR_EXT.1 - Encryption of Sensitive Application Data

FDP_DAR_EXT.1.1	The application shall [*implement functionality to encrypt sensitive data*] in non-volatile memory.

5.2.2.2 FDP_DEC_EXT.1 - Access to Platform Resources

FDP_DEC_EXT.1.1	The application shall restrict its access to [*network connectivity*].
FDP_DEC_EXT.1.2	The application shall restrict its access to [*no sensitive information repositories*].

5.2.2.3 FDP_NET_EXT.1 - Network Communications

FDP_NET_EXT.1.1	The application shall restrict network communication to [*user-initiated communication for [viewing a document, manually refreshing a folder], [automatically syncing for new documents, viewing a sub folder of documents]*].

5.2.2.4 FDP_PRT_EXT.1 - Extended: Protection of Selected User Data

FDP_PRT_EXT.1.1	The TSF shall perform encryption and decryption of the user-selected file (or set of files) in accordance with FCS_COP.1(1).
FDP_PRT_EXT.1.2	The application shall [*implement functionality*] to ensure that all sensitive data created by the TOE when decrypting/encrypting the user-selected file (or set of files) are destroyed in volatile and non-volatile memory when the data is no longer needed.

5.2.3 FMT: Security Management

5.2.3.1 FMT_CFG_EXT.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.
FMT_CFG_EXT.1.2	The application shall be configured by default with file permissions which protect it and its data from unauthorized access.

5.2.3.2 FMT_MEC_EXT.1 - Supported Configuration Mechanism

FMT_MEC_EXT.1.1	The TSF shall [*store and protect configuration options as specified in FCS_COP.1(1)*].

5.2.3.3 FMT_SMF.1 - Specification of Management Functions

FMT_SMF.1.1	The TSF shall be capable of performing the following management functions: [*configure password/passphrase complexity setting;* *change password/passphrase authentication factors;* *[configure if passcode can be skipped if device is configured with passcode and Touch ID;]*].

5.2.4 FPT: Protection of the TSF

5.2.4.1 FPT_AEX_EXT.1 - Anti-Exploitation Capabilities

FPT_AEX_EXT.1.1	The application shall not request to map memory at an explicit address except for [*none*].
FPT_AEX_EXT.1.2	The application shall [*not allocate any memory region with both write and execute permissions*].
FPT_AEX_EXT.1.3	The application shall be compatible with security features provided by the platform vendor.
FPT_AEX_EXT.1.4	The application shall not write user-modifiable files to directories that contain executable files unless explicitly directed by the user to do so.
FPT_AEX_EXT.1.5	The application shall be compiled with stack-based buffer overflow protection enabled.

5.2.4.2 FPT_API_EXT.1 - Use of Supported Services and APIs

FPT_API_EXT.1.1	The application shall use only documented platform APIs.

5.2.4.3 FPT_FEK_EXT.1 - File Encryption Key (FEK) Support

FPT_FEK_EXT.1.1	The TSF shall [*Never store a FEK conditioned from a Password/Passphrase in non-volatile memory*].

5.2.4.4 FPT_KYP_EXT.1 - Extended: Protection of Key and Key Material (FPT_KYP_EXT)

FPT_KYP_EXT.1.1	The TSF shall [*only store keys in non-volatile memory when [wrapped, as specified in FCS_COP.1(5)]*].

5.2.4.5 FPT_LIB_EXT.1 - Use of Third Party Libraries

FPT_LIB_EXT.1.1	The application shall be packaged with only [*Rebar.framework, RebarSupport.framework, libcrypto.dylib, libcurl.dylib, libssl.dylib,*

libswiftCore.dylib, *libswiftCoreGraphics.dylib*, *libswiftCoreImage.dylib*, *libswiftCoreLocation.dylib*, *libswiftDarwin.dylib*, *libswiftDispatch.dylib*, *libswiftFoundation.dylib*, *libswiftObjectiveC.dylib*, *libswiftQuartzCore.dylib*, *libswiftSwiftOnoneSupport.dylib*, *libswiftUIKit.dylib*].

5.2.4.6 FPT_TUD_EXT.1 - Integrity for Installation and Update

FPT_TUD_EXT.1.1	The application shall [*provide the ability*, *leverage the platform*] to check for updates and patches to the application software.
FPT_TUD_EXT.1.2	The application shall be distributed using the format of the platform-supported package manager.
FPT_TUD_EXT.1.3	The application shall be packaged such that its removal results in the deletion of all traces of the application, with the exception of configuration settings, output files, and audit/log events.
FPT_TUD_EXT.1.4	The application shall not download, modify, replace or update its own binary code.
FPT_TUD_EXT.1.5	The application shall [*leverage the platform*] to query the current version of the application software.
FPT_TUD_EXT.1.6	The application installation package and its updates shall be digitally signed such that its platform can cryptographically verify them prior to installation.

5.2.5 FTP: Trusted Path/Channel

5.2.5.1 FTP_DIT_EXT.1 - Protection of Data in Transit

FTP_DIT_EXT.1.1	The application shall [*encrypt all transmitted data with [HTTPS, TLS]*] between itself and another trusted IT product.

5.2.6 Privacy

5.2.5.2 FPR_ANO_EXT.1 - User Consent for Transmission of Personally Identifiable Information

FPR_ANO_EXT.1.1	The application shall [*require user approval before executing [transfer of documents between the TOE and backend data storage]*].

5.2.7 FIA: Identification and Authentication

5.2.7.1 FIA_AUT_EXT.1 - User Authorization

FIA_AUT_EXT.1.1	The application shall [*provide user authorization*] based on [*password/passphrase authorization factors*].

5.2.7.2 FIA_FCT_EXT.1(2) - Extended: User Authorization with Password/Passphrase Authorization Factors

FIA_FCT_EXT.1.1(2)	The TSF shall provide a mechanism as defined in FCS_CKM_EXT.1 and FCS_COP.1(4) to perform user authorization.
FIA_FCT_EXT.1.2(2)	The TSF shall perform user authorization using the mechanism provided in FIA_FCT_EXT.1.1(2) before allowing decryption of user data.

FIA_FCT_EXT.1.3(2)	The TSF shall support the use of multiple instances of authorization factors that result in unique encryption keys.
FIA_FCT_EXT.1.4(2)	The TSF shall verify that the user-entered authorization factors are valid before decrypting the user's encrypted files.
FIA_FCT_EXT.1.5(2)	The TSF shall ensure that the method of validation for each authorization factor does not expose or reduce the effective strength of the KEK, FEK, or CSPs used to derive the KEK or FEK.
FIA_FCT_EXT.1.6(2)	The TSF shall perform user authorization using the mechanism provided in FIA_FCT_EXT.1.1(2) before allowing the user to change the passphrase based authorization factor as specified in FMT_SMF.1(c).

5.2.7.3 FIA_X509_EXT.1 - X.509 Certificate Validation

FIA_X509_EXT.1.1 The application shall [*implement functionality*] to validate certificates in accordance with the following rules:

- RFC 5280 certificate validation and certificate path validation.
- The certificate path must terminate with a trusted CA certificate.
- The application shall validate a certificate path by ensuring the presence of the basicConstraints extension and that the CA flag is set to TRUE for all CA certificates.
- The application shall validate the revocation status of the certificate using [*the Online Certificate Status Protocol (OCSP) as specified in RFC 2560*].
- The application shall validate the extendedKeyUsage field according to the following rules:
    - Certificates used for trusted updates and executable code integrity verification shall have the Code Signing purpose (id-kp 3 with OID 1.3.6.1.5.5.7.3.3) in the extendedKeyUsage field.
    - Server certificates presented for TLS shall have the Server Authentication purpose (id-kp 1 with OID 1.3.6.1.5.5.7.3.1) in the extendedKeyUsage field.
    - Client certificates presented for TLS shall have the Client Authentication purpose (id-kp 2 with OID 1.3.6.1.5.5.7.3.2) in the extendedKeyUsage field.
    - S/MIME certificates presented for email encryption and signature shall have the Email Protection purpose (id-kp 4 with OID 1.3.6.1.5.5.7.3.4) in the extendedKeyUsage field.
    - OCSP certificates presented for OCSP responses shall have the OCSP Signing purpose (id-kp 9 with OID 1.3.6.1.5.5.7.3.9) in the extendedKeyUsage field.
    - Server certificates presented for EST shall have the CMC Registration Authority (RA) purpose (id-kp-cmcRA with OID 1.3.6.1.5.5.7.3.28) in the extendedKeyUsage field.

FIA_X509_EXT.1.2 The application shall treat a certificate as a CA certificate only if the basicConstraints extension is present and the CA flag is set to TRUE.

5.2.7.4 FIA_X509_EXT.2 - X.509 Certificate Authentication

FIA_X509_EXT.2.1	The application shall use X.509v3 certificates as defined by RFC 5280 to support authentication for [*HTTPS, TLS*].
FIA_X509_EXT.2.2	When the application cannot establish a connection to determine the validity of a certificate, the application shall [*not accept the certificate*].

5.3 TOE Security Assurance Requirements

The security assurance requirements in Table PP_APP_SW_SAR are included in this ST by reference from the PP APP SW and PP APP SWFE.

PP_APP_SW_SAR

Requirement Class	Requirement Component
ADV: Development	ADV_FSP.1 Basic Functional Specification
AGD: Guidance Documentation	AGD_OPE.1 Operational User Guidance
	AGD_PRE.1 Preparative Procedures
ALC: Life-cycle Support	ALC_CMC.1 Labeling of the TOE
	ALC_CMS.1 TOE CM Coverage
	ALC_TSU_EXT.1 Timely Security Updates
ATE: Tests	ATE_IND.1 Independent Testing – Conformance
AVA: Vulnerability Assessment	AVA_VAN.1 Vulnerability Survey

These assurance requirements imply the following requirements from CC class ASE: Security Target Evaluation.

- ASE_CCL.1 Conformance claims
- ASE_ECD.1 Extended components definition
- ASE_INT.1 ST introduction
- ASE_OBJ.1 Security objectives for the operational environment
- ASE_REQ.1 Stated security requirements
- ASE_TSS.1 TOE summary specification

Consequently, the assurance activities specified in PP APP SWFE apply to the TOE evaluation.

5.4 Superseded/Combined Requirements

The security assurance requirements in Table PP_APP_SW_ SUPERSEDED that are referenced below are superseded or combined by requirements in the PP APP SWFE Protection Profile.

PP_APP_SW_SUPERSEDED

Requirement Class	Requirement Component
FCS: Cryptography Support	FCS_COP.1(1) - Cryptographic Operation - Encryption/Decryption
	FCS_COP.1(4) - Cryptographic Operation - Keyed-Hash Message Authentication
FMT: Security Management	FMT_SMF.1 - Specification of Management Functions

6. TOE Summary Specification

This chapter describes the security functions for:

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

6.1 Cryptographic Support

The TOE makes use of both cryptographic functions of the platform (iOS) and Rebar. Rebar provides a second layer of encryption for both DAR and DIT via its cryptographic implementation. Where applicable the TOE will invoke cryptographic functionality, such as marking files as NSFileProtectionComplete and invoking the system provided DRBG functionality to seed Rebar's DRBG. Rebar's implementation of cryptographic functionality enables signing, hashing, key-hash message authentication, key generation, and symmetric encryption.

6.1.1 FCS_CKM.1(1) - Cryptographic Asymmetric Key Generation

The TOE does not generate RSA keys as it is not required for TLS sessions as the TOE acts as a receiver of keys from the server. The TOE does generate ECC keys in compliance with the DSS standard.

6.1.2 FCS_CKM.1(2) - Cryptographic Symmetric Key Generation

Rebar leverages the implemented DRBG functionality to generate all symmetric keys. Keys are generated of a length of 32 bytes and IV values are generated of a length of 16 bytes.

6.1.3 FCS_CKM.2 - Cryptographic Key Establishment

Rebar's implementation for TLS connectivity leverages Elliptic curve-based key establishment schemes and RSA key establishment schemes. The TOE does not generate RSA keys and only receives RSA keys.

6.1.4 FCS_CKM_EXT.1 - Cryptographic Key Generation Services (PP APP SW)

The TOE generates ECC keys in compliance with the DSS standard.

6.1.5 FCS_CKM_EXT.1 - Key Encrypting Key (KEK) Support (PP APP SWFE)

Rebar leverages PBKDF2 with HMAC-SHA-256 at 4096 iterations and an output key size of 256 bits with the user supplied passphrase to use the app. The user will supply a passphrase to the application. The application will feed that key into the PBKDF2 function using HMAC-SHA-256 leveraging 4096 iterations. The output will be a 256 bit key that is the KEK.

The salt leverages to generate the KEK with PBKDF2 is generates with the compliant RBG function and stored within the device keychain.

The app can be configured to be run without the use of a passphrase when the app is opened. *This is not a part of the TOE*. When this option is configured, devices with the Touch ID/Secure Enclave store a ECC value in the Secure Enclave. A 256-bit key is generated and stored in the system keychain. That key is signed with the ECC key from the Secure Enclave. That value is fed to the PBKDF2 function to derive a key.

6.1.6 FCS_CKM_EXT.1(A) Extended: Cryptographic Key Generation (Password/Passphrase Conditioning)

Rebar leverages PBKDF2 with HMAC-SHA-256 at 4096 iterations and an output key size of 256 bits with the user supplied passphrase to use the app.

Salt is generated using the DRBG (defined under FCS_RBG_EXT.1) functionality within the TOE to salt the PBKDF2 function, the salt is 128 bits in length.

6.1.7 FCS_CKM_EXT.2 - Cryptographic key generation (FEK)

Rebar's implementation relies on both FEK encryption keys generated with RBG functionality and password/passphrase.

Rebar's RBG functionality conforms to the FCS_RBG_EXT.1 specification for RBG derived values. Rebar generates keys of 256 bits from the RBG. The FEK is handled in two ways. Rebar stores its secure settings in a AES 256 encrypted database Rebar Management Database (RMD). This database is encrypted with a 256 bit key that is derived from a salt (128 byte) and user provided passcode. This value generates a key leveraging PBKDF2 functionality with HMAC256 hashing (Our Database Encryption Key DEK).

If the TOE generates other databases, a RBG generated 256-bit key is generated and stored within the RMD. When the databases are unlocked, the stored key for other databases is available to decrypt the other databases.

Files stored locally on the device are encrypted with a FEK that is generated from the RBG with a key length of 256 bits. These FEK are also stored within the RMD.

The RMD FEK is kept in memory long enough to facilitate decrypting the RMD. Once the database has been decrypted, the memory space in volatile memory is erased. In the even the user wishes to change the passcode for the app, the RMD FEK is HMAC512 hashed and stored within the RMD. When the user prompts to change the RMD FEK, the hashed values are compared.

6.1.8 FCS_CKM_EXT.4 - Extended: Cryptographic Key Destruction

Rebar's implementation retrieves all keys in a binary (Data within iOS) format. The FEKs are protected with a KEK derived from the user’s passcode. After the FEK have been decrypted with the KEK, the KEK is destroyed in volatile memory.

When the user exits the application for a configurable period of time (up to 3 minutes to immediately) the TOE will destroy the FEK loaded into memory.

To destroy the keys loaded into volatile memory, the keys are overwritten with 0's. This is a low-level operation performed by iterating the mutable binary data and setting the values to the 0-bit value. Then the keys are set to nil which enables the platform to release the memory.

For non-volatile storage, where the encrypted FEKs are stored, Rebar's implementation will perform a "shred" operation on files that are to be deleted. This operation is performed on any file deleted from the file system as well as the RMD when the RMD is deleted. This ensures that any key material is securely deleted during the deletion of the files or the RMD. The shred operation generates random data from the DRBG source and will overwrite a file with the random data collected from the DRBG.

REBAR_KEYS

Key	Description
KEK	The KEK is a user conditioned key that is derived from the user entered passcode through PBKDF2 and used to decrypt the FEK that are used by the system. Each FEK is encrypted by the KEK using AES Key Wrapping. This key is destroyed with zero overwriting in memory once the individual FEK are decrypted and loaded into memory.
RMD FEK	The RMD FEK is a file encryption key that is used to encrypt the RMD. This is a combined key of the KEY and IV, which is split when used. The RMD KEK is generated when the user initially authenticates with the app leveraging their password. The KEK is loaded into volatile memory when the user authenticates with their passcode for the app. When the user's session times out, the KEK is written over with zeros in memory.
Individual Database FEK	The Individual Database FEK is a file encryption key that is used to encrypt the databases other than the RMD. This is a combined key of the KEY and IV, which is split when used. The Individual Database KEK is generated when the user initially authenticates with the app leveraging their password. The KEK is loaded into volatile memory when the user authenticates with their passcode for the app. When the user's session times out, the KEK is written over with zeros in memory.
File FEK	The File FEK is a file encryption key that is used to encrypt files stored locally on the file system for the TOE. This is a combined key of the KEY and IV, which is split when used. The File FEK is generated when the user initially authenticates with the app leveraging their password. The KEK is loaded into volatile memory when the user authenticates with their passcode for the app. When the user's session times out, the KEK is written over with zeros in memory.

6.1.9 FCS_COP.1(1) - Cryptographic operation (Data Encryption)

Rebar's implementation of data at rest encryption leverages AES 256 exclusively. Rebar's RMD is encrypted with AES CBC 256 encryption leveraging PBKDF2 derived encryption keys. Other databases generated and managed by Rebar are using AES CBC 256 encryption keys (generated by the implemented RBG).

Files (other than RMD and other databases) are stored leveraging AES CBC 256 encryption and 128 bit IV values.

All AES 256 encryption is leverages using the CBC cryptographic algorithms for data at rest.

The TOE additionally has AES GCM 256 and AES CBC 128 for TLS operations/data in transit.

This SFR supersedes the APP_SW SFR for Cryptographic Operation (Data Encryption) and leverages the relevant technical decision including GCM as a supported cipher. While GCM was supported in APP_SW 1.2, it was lacking from APP_SWFE.

6.1.10 FCS_COP.1(2) - Cryptographic Operation - Hashing

Rebar's implementation implements hashing with SHA-1, SHA-256, SHA-384 and SHA-512 for all cryptographic operations that are performed.

6.1.11 FCS_COP.1(3) - Cryptographic Operation – Signing

Rebar's implementation for RSA and ECDSA signature generation and verification conform to FIPS 186-4.

6.1.12 FCS_COP.1(4) - Cryptographic Operation (Keyed-Hash Message Authentication)

HMAC-SHA-256 and HMAC-SHA-512 services are available via the Rebar platform. The app users HMAC-SHA-256 hashing to sign HTTPS requests with the RTX signing token.

This SFR supersedes the APP_SW SFR for Cryptographic Operation (Keyed-Hash Message Authentication). This requirement is more specific and specifically calls out the distinction of platform provided HMAC vs implemented HMAC.

6.1.13 FCS_COP.1(5) - Cryptographic operation (Key Wrapping)

Rebar manages a keychain for protecting encrypted assets on the file system, a set of File Encryption Key's (FEK) that encrypt and decrypt files on the file system. Each FEK will include the AES 256 bit Key and 128 bit IV.

Rebar maintains at a minimum 3 FEKs for different files.

Rebar Management Database (RMD) FEK: A key used to encrypt/decrypt the RMD for securely storing settings and configuration times
Individual Database FEK: A key used to encrypt each of the individual databases the app may create and manage
File FEK: A key used for encryption and decryption of files that maybe stored on the file system (Considered a "SET" of files managed by the FEK) generated by the app.

To protect these keys, Rebar leverages *AES Key Wrap (256 bit)* to wrap and protect the keys conforming to " *NIST SP 800-38F" for Key Wrap (section 6.2) and Key Wrap with Padding (section 6.3)*".

The FEK is protected by a KEK derived from FCS_CKM_EXT.1 - a password based key derived based one requirements laid out in FCS_CKM.1(A) and FIA_FCT_EXT.1(2).

FEK Key Wrapping Workflow:

User enters passcode into app.
App leverages PBKDF2 with HMAC-SHA-256 and generates a KEK.
If no FEKs have been generated, conforming RBG will generate a 256 bit key and 128 bit IV. RKM will AES Key Wrap the FEKSs and store them in nonvolatile memory with the KEK.

When reading the FEK protected by the KEK:
RKM will read the RKM file into memory and AES Key Wrap unwrap the FEKs with the provided KEK.
The KEK will be destroyed in memory and the FEK will be made available to application resources.
The FEK will be kept in volatile memory and destroyed on the application timeout window (e.g. 5 minutes of the user exiting the app).
The FEK will be unwrapped again when the user is prompted to enter their passcode to condition the KEK.

6.1.14 FCS_HTTPS_EXT.1 - HTTPS Protocol

Rebar implements RFC 2818 and leverages TLS 1.2 for DIT conformance.

6.1.15 FCS_IV_EXT.1 - Extended: Initialization Vector Generation

Rebar's implementation derives IV 128 bit values leveraging RBG referenced in FCS_RBG_EXT.1.1. This leverages Cipher Block Chaining (CBC) required in Appendix G.

When the RKM generates a FEK, it will generate a key of a 256 bits and an IV of 128 bits. This FEK will be encrypted with the KEK and stored with the RKM.

6.1.16 FCS_KYC_EXT.1 - Key Chaining and Key Storage

The TOE implements a Keychain and Key Storage leveraging AES Key Wrapping to protect the keys. The FEK are protected with a KEK that is derived from the user provided passphrase that is conditioned into a KEK using PBKDF2.

At no point can the chain be broken without a cryptographic exhaust or knowledge of the KEK or FEK and the effective strength of the FEK is maintained throughout the Key Chain as specified in the requirement.

6.1.17 FCS_RBG_EXT.1 - Random Bit Generation Services

Rebar's implementation of DRBG functionality is implemented within the TOE using FIPS Validated DRBG and seeded with platform FIPS Validated CTR_DRBG algorithms (noted in https://www.apple.com/business/docs/iOS_Security_Guide.pdf). This implementation conforms to the NIST Special Publication 800-90A requirements.

To seed the TOE, the TOE pulls 4160 bits of entropy to seed from the platform DRGB functionality with a reseed interval of 2^24.

The TOE CTR_DRBG random function is seeded on iOS with the SecRandomCopyBytes platform DRBG functionality as required by the SFR.

To facilitate this functionality for our FIPS Validated OpenSSL calls, Monkton has implemented required function callbacks in the OpenSSL FIPS User Guide: https://www.openssl.org/docs/fips/UserGuide-2.0.pdf. These callbacks allow Monkton so specify the entropy source and reseed rates to provide entropy from the platform for internal OpenSSL cryptography calls requiring DRBG methods.

6.1.18 FCS_RBG_EXT.2 - Random Bit Generation from Application

Rebar's implementation of CTR_DRBG is AES-256. The deterministic RBG shall be seeded by an entropy source that accumulates entropy from a platform-based DRBG and no other noise source with a minimum of 256 bits of entropy at least equal to the greatest security strength (according to NIST SP 800-57) of the keys and hashes that it will generate.

6.1.19 FCS_STO_EXT.1 - Storage of Secrets

Rebar's implementation leverages secure non-volatile storage backed by AES-256 encryption to store server authentication tokens and file encryption keys for files stored locally on the device. The Rebar "Keystore" is conditioned with salt that is securely stored within the platform, a user provided passphrase, and conditioned with a PBKDF2 function leveraging HMAC-SHA-256.

6.1.20 FCS_TLSC_EXT.1 - TLS Client Protocol

Rebar's implementation allows the configuration of the cipher suites within the Rebar App configuration file. The cipher suites in addition to TLS_RSA_WITH_AES_128_CBC_SHA are provided as part of the app configuration during deployment. They can also be modified leveraging Mobile Device Management configuration with the Managed App Configuration settings.

These settings enable the developer or administrator to change the acceptable cipher suites to the needs of the deployment configuration for relevant security configurations. The supported ciphers are listed in Table REBAR_CIPHER_SUITES.

REBAR_CIPHER_SUITES

Cipher Suite	Supported
TLS_RSA_WITH_AES_128_CBC_SHA	Yes
TLS_RSA_WITH_AES_256_CBC_SHA256	Yes
TLS_RSA_WITH_AES_256_GCM_SHA384	Yes
TLS_DHE_RSA_WITH_AES_128_CBC_SHA256	No
TLS_DHE_RSA_WITH_AES_256_CBC_SHA256	No
TLS_DHE_RSA_WITH_AES_256_GCM_ SHA384	No
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	Yes
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	Yes
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	Yes
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	Yes
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	Yes
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	Yes
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384	Yes
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	Yes

The requested URL provided to connect to a server will be validated against the provided certificate. The application will extract the SAN names from the requested certificate. The potential SAN names will be validated against the requested URL. The TOE specifically supports checking the associated SAN names in the certificate only, as Common Names have been recommended as deprecated in RFC6125 (https://tools.ietf.org/html/rfc6125). The TOE only enforces URLs, not URNs. Wildcards are supported as well as IP Address (IP addresses will be deprecated in the future).

6.1.21 FCS_TLSC_EXT.3 - TLS Client Protocol

Rebar implements the requisite supported signature algorithms for SHA message hashing. The TOE has selected the non-configurable signature algorithms of:

REBAR_SIGNATURE_ALGORITHMS

Elliptical Curve	Supported
ECDSA+SHA256	Yes
ECDSA+SHA384	Yes
ECDSA+SHA512	Yes
RSA+SHA256	Yes
RSA+SHA384	Yes
RSA+SHA512	Yes

6.1.22 FCS_TLSC_EXT.4 - TLS Client Protocol

Rebar implements the ECDHE required signature algorithms to support message signing for TLS requiring ECDHE. The TOE has selected the non-configurable curves of:

REBAR_TLS_CURVES

Elliptical Curve	Supported

P-521/secp521r1	Yes
P-384/secp384r1	Yes
P-256/secp256r	Yes

6.2 User Data Protection

The TOE requests no platform resource or hardware permissions from the user on iOS. Network connections are initiated in a variety of ways. Users tapping on files begins the download process for files. When the user opens the app or at internal intervals the app will request a refresh from the server for the files that have changed or directory contents. When a user opens a sub folder, the files from the sub folder will be loaded automatically.

6.2.1 FDP_DAR_EXT.1 - Encryption of Sensitive Application Data

The TOE, by being built with Rebar implement its own functionality to encrypt data, in addition to leveraging the OS approved full disk encryption. Rebar implements the Extended Package for Software File Encryption Version 1.0 to provide encryption of files, configuration items, and other sensitive data that is stored within the application itself.

The TOE leverages the NSFileProtectionComplete flag as an entitlement, meaning that while the device is powered off or locked, data within the app remains encrypted. Further information on the NSFileProtectionComplete configuration can be obtained in the Apple Security Guide: https://www.apple.com/de/business/docs/iOS_Security_Guide.pdf

6.2.2 FDP_DEC_EXT.1 - Access to Platform Resources

The TOE only accesses network resources to operate. The TOE accesses no sensitive repositories.

6.2.3 FDP_NET_EXT.1 - Network Communications

The TOE has network initiated actions that are caused by both the user and the application itself while running. The user initiates refreshes of content, login, and downloading of content from the remote server. The app can initiate refreshes of content (files, documents) without user interaction.

6.2.4 FDP_PRT_EXT.1 - Extended: Protection of Selected User Data

Rebar's implementation complies with FCS_COP.1(1) for encryption and decryption of all user files. The FEKs are managed by the RKM. The FEKs are loaded into memory by the RKM when the user enters their passcode and a KEK is conditioned from the passcode. The KEK is used to unwrap the FEKs.

FEKs are used to encrypt and decrypt data stored in non-volatile memory. The FEKs are stored encrypted in non-volatile memory with the KEK.

To manage memory of objects decrypted with FEKs, Monkton leverages both platform cleanup of resources and its own implementation of cleanup of resources.

When an encrypted physical file is deleted from the TOE, Rebar will obtain a file handle to the file and overwrite the files with random data. While the neither NIAP Protection Profile requires this action (Except for FCS_CKM_EXT.4 which handles key destruction), Monkton believes this is a best practice. Monkton then leverages the platform file deletion functionality to cleanup resources.

In the instance where the TOE must write an unencrypted file to non-volatile storage – Monkton will delete the file from non-volatile storage and perform the same FCS_CKM_EXT.4 compliant procedure to ensure the file is

deleted securely. This can occur when the TOE may need to render a PDF in the platforms native PDF rendering tools. If the user exits the app or the user closes the document the file will be deleted immediately. The TOE makes a best effort to stream, in memory, the files in a decrypted format to keep them from being written encrypted to non-volatile memory.

For databases managed with the TOE, Monkton has implemented a secure “scratch space” for data to be decrypted in. This “scratch space” is a byte array that allows the TOE to securely zeroize data when the TOE performs a cleanup operation when the databases are closed. This ensures that data stored within the databases are limited to exposure when data is loaded into memory.

Temporary files and resources may be created, but are generally not while the app is running. The database may create a log file, but while on disk that log file is encrypted. In the instances where a file is viewed and a temporary file is created unencrypted (By the TOE encryption), the file will be deleted when the user stops viewing the file. If the TOE exits through a forced exit, the file will be deleted on the next opening.

When the RKM has the KEK changed, the KEK will be used to re-encrypt the FEK contained within the RKM. In the event a FEK changes for a file or set of files, the files will be re-encrypted with the new FEK and the old files will be deleted.

6.3 Security Management

6.3.1 FMT_CFG_EXT.1 - Secure by Default Configuration

By default, the TOE is configured to be secure.

To use the TOE, the user must provide a username and password.

The application does not install with default credentials. When a user is not logged in the TOE contains no data stored within it.

6.3.2 FMT_MEC_EXT.1 - Supported Configuration Mechanism

Certain configuration attributes, such as the RTX are stored within the RMD, which is protected by the RKM, which is protected with the user derived KEK and FEK. For elements stored within the RMD, they are protected with FCS_COP.1(1) that leverages AES Key Wrapped FEK protected by the password derived KEK.

6.3.3 FMT_SMF.1 - Specification of Management Functions

Rebar's implementation allows for the end user to change their passcode. The passcode policy is set at the administrator level, as part of the OE, but not part of the TOE boundary, and delivered as policy via TLS for consumption by the TOE.

Within the app, users are able to modify their passcode via the "Change Passcode" screen found in the Settings screen. Users will be prompted to enter their current passcode, then enter a new passcode and confirm the new passcode. The current passcode is validated to be correct by generating a key via PBKDF2 and compared against the SHA512 hash of the key, which is stored in the RMD. If the new passcode satisfies complexity requirements, the passcode will be run through the PBKDF2 function with a salt, a 16 byte random value, which is generated via the implemented DRBG functionality, and the keychain KEK will be updated, re-encrypting each FEK in the keychain.

The users KEK will then be hashed using a SHA512 hashing algorithm and stored within the RMD. This hashed value is used to perform the validation that the "Current Passcode" is correct. (The "Change Passcode" KEK is hashed with the SHA512 algorithm and compared to the RMD stored value).

The complexity requirements are configured via the OE. The configuration options allow the administrator to set a minimum passphrase length and character requirements which are delivered as policy via TLS. The user can be required to enter a minimum set of characters.

Once set, the complexity policy is downloaded by the mobile app over TLS for when the user sets their passphrase, either initially or as an update, enforcing the passphrase policy.

It is worth noting that this SFR is combined with the APP_SW and APP_SWFE. This is due to the specific language in the relevant Technical Decision for FMT_SMF.1. More so, the SFR does not contain any selections for the base PP, but does include selections for the APP_SWFE.

6.4 Protection of the TSF

6.4.1 FPT_AEX_EXT.1 - Anti-Exploitation Capabilities

The TOE does not implement any mmap/mprotect calls. Additionally, the TOE and binaries compiled by Monkton are compiled with the -fstack-protector-strong set.

6.4.2 FPT_API_EXT.1 - Use of Supported Services and APIs

The TOE only uses supported platform APIs. Leveraged APIs are documented in Appendix APIs.

6.4.3 FPT_FEK_EXT.1 - File Encryption Key (FEK) Support

Rebar’s implementation of key management does not store the FEK in a decrypted format in non-volatile memory at any point in time. The RKM only writes encrypted FEK to the file system for the keychain that is managed by the RKM. The RKM leverages the KEK conditioned from the users provided passphrase.

The FEKs are wrapped and unwrapped using AES Key Wrapping using the KEK as the key to perform the cryptographic operation. The RKM will store the FEK type (RMD, Individual Database, File) as metadata for the specific FEK. The FEK will be AES Key Wrapped with the KEY and IV combined into a single value.

The RKM provides three FEKs for usage by the TOE. The RMD FEK, the Individual Database FEK and the File KEK. Each of these are denoted by the identifier associated with the FEK that is stored in non-volatile memory.

6.4.4 FPT_KYP_EXT.1 - Extended: Protection of Key and Key Material (FPT_KYP_EXT)

Rebar's implementation generates and stores keys in the RMD, where the RMD is encrypted and protected with a passcode derived FEK. The keys are generated using RBG to the required bit standards (256 bit) and stored within the RMD - available only after decrypted and unavailable when the database is encrypted.

6.4.5 FPT_LIB_EXT.1 - Use of Third Party Libraries

Rebar's implementation by default integrates several Open Source packages to build and deliver some key components of Rebar. Rebar provides a means for developers to leverage the required licensing language from these libraries in a "legal" screen within the app, and to add their own. Rebar includes the following OSS software referenced in the ST.

6.4.6 FPT_TUD_EXT.1 - Integrity for Installation and Update

The TOE is distributed as a signed IPA file with either Monkton Inc's Enterprise Apple Signing Certificate or Monkton Inc's App Store Apple Signing Certificate. Users obtain updates to the TOE either through the Apple App Store (B2B), an MDM managed distribution mechanism, or another enterprise distribution mechanism. iOS will only install the app if the signing certificates are valid.

The Settings tab in the app provides the user the app version identifier.

In the event of a disclosure of a vulnerability found in the TOE, Monkton will provide an update to the TOE within 45 calendar days or sooner depending on the severity of the issue discovered. Monkton can accept vulnerability reports via our website https://monkton.io or emailing security@monkton.io to report any security related issues.

6.5 Trusted Path/Channel

6.5.1 FTP_DIT_EXT.1 - Protection of Data in Transit

Rebar's implementation forces all Web API calls to be performed with HTTPS over TLS 1.2. This is enforced at several layers within the application. Rebar provides an easy to use interface for developers to perform simple HTTP GET/PUT/POST methods to the Web API server.

6.6 Privacy

6.6.1 FPR_ANO_EXT.1 - User Consent for Transmission of Personally Identifiable Information

The TOE may transfer PII dependent on which content is being accessed via the content stored within the cloud. To be compliant with the PP requirements of potentially transferring PII, users will be prompted after initial authentication with the app to notify them that the application may transmit PII. This will ensure that the user understands the TOE may transmit PII but it is not guaranteed to.

6.7 Identification and Authentication

6.7.1 FIA_AUT_EXT.1 - User Authorization

Users are authorized to the application using a passphrase that is entered when the user logs into the application. This authorization step will perform the requisite PBKDF2 key generation. Once the key has been conditioned it will be SHA512 hashed and validated against a value stored in the device keychain. If the values match, the TOE will attempt to open the RMD. If the RMD is opened and can be read from successfully the user is considered authenticated.

6.7.2 FIA_FCT_EXT.1(2) - Extended: User Authorization with Password/Passphrase Authorization Factors

Users are authorized to the application using a passphrase that is entered when the user logs into the application. This authorization step will perform the requisite PBKDF2 key generation.

Once the key has been conditioned as a KEK from the passphrase, it will be used to unlock the RKM. If the RKM fails to decrypt, it will indicate that the KEK was not valid and indicate that to the user. If the KEK is correct, the RKM will be unlocked, the FEKs will be provides to the TOE, and the KEK will be destroyed.

Upon successful entry of the passphrase and validation, the RMD will be opened.

When the KEK is set, it will then be hashed with SHA512 and stored within the secure storage. This will be used at later times to enable the user to change the KEK for the RKM (with a new passphrase for instance). When the user attempts to change the passphrase that the KEK is derived from, the user will enter the existing passphrase, which

will condition a KEK from the PBKDF2 algorithm, which is then hashed with SHA-512 and compared to the value stored within secure storage. If the values match, the new passcode will be used to condition a KEK using PBKDF2 and re-encrypt the RKM with the newly provided KEK. Additionally, the TOE will update the cached SHA-512 value of the new KEK enabling the user to change the passphrase in the future.

Once the passphrase has been changed, the passphrase will no longer be able to decrypt the RKM. Additionally, due to salting, even if the passphrase is exactly the same, the newly derived KEK will not be able to decrypt the RKM.

6.7.3 FIA_X509_EXT.1 - X.509 Certificate Validation

Rebar's implementation for certificate validation implements functionality to validate the chain of certificates, requisite OID and certificate flags are embedded in the certificates. Additionally, Rebar implements SSL Pinning, leveraging SHA-512 hashes of the certificates to enable pinning of any certificate on the stack.

When the TOE attempts to make a connection to the server and receives the certificate for the handshake after "server hello" handshake. Once this certificate is received, the application will validate the certificate chain, SSL Pinning, and validate the required attributes are present in the certificate chain.

The TOE evaluates each of the certificates in the chain. The TOE will check OCSP to ensure that the child certificate has not been revoked. The intermediate certificates and the root certificates are validated against the embedded certificate chain file that is loaded into the TOE. Each of the roots and intermediates are validated to be in the chain. Once those checks are completed, the TOE will evaluate the required attributes exist within the root and intermediate certificates.

Rebar relies on device validation of updates, to which the check for *id-kp 3 with OID 1.3.6.1.5.5.7.3.3* is not applicable.

Rebar does not enable client authentication at this time, to which the check for *id-kp 2 with OID 1.3.6.1.5.5.7.3.2* is not applicable.

Rebar does not facilitate SMIME signing, to which the check for *id-kp 4 with OID 1.3.6.1.5.5.7.3.4* is not applicable.

Rebar does not facilitate EST, to which the check for *id-kp-cmcRA with OID 1.3.6.1.5.5.7.3.28* is not applicable.

Rebar does implement OCSP checks. Due to the fact that OCSP signing is not widely implemented, *id-kp 9 with OID 1.3.6.1.5.5.7.3.9* compliance is delegated with the configuration flag rebar.continueOnOCSPError being set to FALSE. The OCSP Signing check will fail in a majority of cases due to OCSP Signing OID not being leveraged.

6.7.4 FIA_X509_EXT.2 - X.509 Certificate Authentication

Rebar's implementation supports only X509v3 certificates for TLS & HTTPS connections. Additionally, Rebar's implementation allows for policies to be set in the Rebar Application Configuration file that allow for the SSL Certificate to be allowed of OCSP checks fail for the certificate.

The TOE will only allow certificates to be used that conform to the validation rules defined in FIA_X509_EXT.1. This includes defined rules in the Application Protection Profile as well as SSL Pinning implemented in the TOE as an additional security check.

Failure to validate (excusing the optional OCSP Error continuation) will prevent the certificate from being used and the connection to be rejected by the TOE. The user will be presented a message that certificate validation failed and thus the connection has failed.

Subsequent requests will still attempt to validate the certificate and if the certificate continues to fail the connection will be denied.

7. Protection Profile Claims

7.1 PP APP SW

This ST conforms to the Protection Profile for Application Software, Version 1.2, (PP APP SW) As explained in Security Problem Definition, the Security Problem Definition of the PP APP SW has been included by reference into this ST.

As explained in Security Objectives (PP APP SW), the Security Objectives of the PP APP SW have been included by reference into this ST.

The following table identifies all the security functional requirements in this ST. Each SFR is reproduced from the PP APP SW and operations completed as appropriate.

CLAIMS_PP_APP_SW

Requirement Class	Requirement Component	Source
FCS: Cryptographic support	FCS_CKM.1(1) Cryptographic Asymmetric Key Generation	PP APP SW
	FCS_CKM.1(2) - Cryptographic Symmetric Key Generation	PP APP SW
	FCS_CKM.2 - Cryptographic Key Establishment	PP APP SW
	FCS_COP.1(2) - Cryptographic Operation - Hashing	PP APP SW
	FCS_COP.1(3) - Cryptographic Operation – Signing	PP APP SW
	FCS_HTTPS_EXT.1 - HTTPS Protocol	PP APP SW
	FCS_RBG_EXT.1 - Random Bit Generation Services	PP APP SW
	FCS_RBG_EXT.2 - Random Bit Generation from Application	PP APP SW
	FCS_STO_EXT.1 - Storage of Secrets	PP APP SW
	FCS_TLSC_EXT.1 - TLS Client Protocol	PP APP SW
	FCS_TLSC_EXT.3 - TLS Client Protocol	PP APP SW
	FCS_TLSC_EXT.4 - TLS Client Protocol	PP APP SW
FDP: User Data Protection	FDP_DAR_EXT.1 - Encryption of Sensitive Application Data	PP APP SW
	FDP_DEC_EXT.1 - Access to Platform Resources	PP APP SW
	FDP_NET_EXT.1 - Network Communications	PP APP SW
FIA: Identification and Authentication	FIA_X509_EXT.1 - X.509 Certificate Validation	PP APP SW
	FIA_X509_EXT.2 - X.509 Certificate Authentication	PP APP SW
FMT: Security Management	FMT_CFG_EXT.1 - Secure by Default Configuration	PP APP SW
	FMT_MEC_EXT.1 - Supported Configuration Mechanism	PP APP SW
FPT: Protection of the TSF	FPT_AEX_EXT.1 - Anti-Exploitation Capabilities	PP APP SW
	FPT_API_EXT.1 - Use of Supported Services and APIs	PP APP SW
	FPT_LIB_EXT.1 - Use of Third Party Libraries	PP APP SW
	FPT_TUD_EXT.1 - Integrity for Installation and Update	PP APP SW
FTP: Trusted Path/Channel	FTP_DIT_EXT.1 - Protection of Data in Transit	PP APP SW
Privacy	FPR_ANO_EXT.1 - User Consent for Transmission of Personally Identifiable Information	PP APP SW

7.2 PP APP SWFE

This ST conforms to the Extended Package for Software File Encryption Version 1.0, Version 1.0, (PP APP SWFE) As explained in Security Problem Definition, the Security Problem Definition of the PP APP SWFE has been included by reference into this ST.

As explained in Security Objectives (PP APP SWFE), the Security Objectives of the PP APP SWFE have been included by reference into this ST.

The following table identifies all the security functional requirements in this ST. Each SFR is reproduced from the PP APP SWFE and operations completed as appropriate.

CLAIMS_PP_APP_SWFE

Requirement Class	Requirement Component	Source
FCS: Cryptographic Support	FCS_CKM.1(A) - Cryptographic key generation (Password/Passphrase conditioning)	PP APP SWFE
	FCS_CKM_EXT.1 - Key Encrypting Key (KEK) Support	PP APP SWFE
	FCS_CKM_EXT.2 - Cryptographic key generation (FEK)	PP APP SWFE
	FCS_CKM_EXT.4 - Extended: Cryptographic Key Destruction	PP APP SWFE
	FCS_COP.1(1) - Cryptographic operation (Data Encryption)	PP APP SWFE
	FCS_COP.1(4) - Cryptographic Operation (Keyed-Hash Message Authentication)	PP APP SWFE
	FCS_COP.1(5) - Cryptographic operation (Key Wrapping)	PP APP SWFE
	FCS_IV_EXT.1 - Extended: Initialization Vector Generation	PP APP SWFE
	FCS_KYC_EXT.1 - Key Chaining and Key Storage	PP APP SWFE
FDP: User Data Protection	FDP_PRT_EXT.1 - Extended: Protection of Selected User Data	PP APP SWFE
FIA: Identification and Authentication	FIA_AUT_EXT.1 - User Authorization	PP APP SWFE
	FIA_FCT_EXT.1(2) - Extended: User Authorization with Password/Passphrase Authorization Factors	PP APP SWFE
FMT: Security Management	FMT_SMF.1 - Specification of Management Functions	PP APP SWFE
FPT: Protection of the SF	FPT_FEK_EXT.1 - File Encryption Key (FEK) Support	PP APP SWFE
	FPT_KYP_EXT.1 - Extended: Protection of Key and Key Material (FPT_KYP_EXT)	PP APP SWFE

Appendix: APIs

- Rebar.framework
- RebarSupport.framework
- libcrypto.dylib
- libcurl.dylib
- libssl.dylib
- libswiftCore.dylib
- libswiftCoreGraphics.dylib
- libswiftCoreImage.dylib
- libswiftCoreLocation.dylib
- libswiftDarwin.dylib
- libswiftDispatch.dylib
- libswiftFoundation.dylib
- libswiftObjectiveC.dylib
- libswiftQuartzCore.dylib
- libswiftSwiftOnoneSupport.dylib
- libswiftUIKit.dylib

Appendix: Copyright Notices