itsme Second Factor Attestation Engine Security Target

Version 1.4.8
42 pages
24/10/2025

Content

List of Figures

List of Tables

1 ST introduction

1.1 ST reference

Title: itsme Second Factor Attestation Engine Security Target
Version: 1.4.8
Publication date: 24/10/2025

1.2 TOE reference

TOE identification: itsme Second Factor Attestation Engine
TOE version: 1.0.0

Note that the TOE version is the same for both the client and server subsystem software packages (see section 1.4.2).

1.3 TOE overview

The TOE allows a customer's application to securely delegate validation and management of an enrolled user's second factor, which can either be a PIN or biometrics. After validation, an attestation for use with a third party is provided.

The TOE consists of a client and server subsystem:

- The TOE client subsystem securely handles the user's PIN and biometrics. Biometrics are handled indirectly through the environment with the support of the mobile device's SE/TEE.
- The TOE client subsystem allows to register additional keys (called 'subject' keys) linked to the user with the server. Note that the generation, maintenance and actual usage of these keys is out of scope for the TOE. See section 1.4.1.1 for an example showing how these keys can be used along with the TOE.
- The TOE server subsystem allows for the secure enrolment, verification and updates of the user's second factor (PIN or biometrics) and subject keys.
- The TOE server subsystem produces an attestation (including a reference to all registered subject keys for that user) and audit records of the correct usage of the second factor.

The TOE client subsystem handles 4 types of cryptographic key pairs:

- Client key pair: to establish an identifier for the user;
- PIN key pair: to verify the PIN of the user;
- Biometrics key pair (protected by the mobile device's SE/TEE): to verify the biometrics of the user; and
- Subject key pair(s): a key pair related to the user for usage outside of the TOE scope. By registering these public keys with the TOE, the attestation that is delivered by the TOE is linked to these keys.

The TOE client subsystem needs to register the above public keys with the TOE server subsystem. During initial enrolment, the TOE client subsystem registers the *ClientPublicKey*, at least one of *PINPublicKey* and *BiometricPublicKey*, and optionally *SubjectPublicKey*(s). For every subsequent update to the registered keys at the TOE server subsystem, the user will first need to verify using their *ClientPrivateKey* and a registered second factor.

1.3.1 Usage and major security features of the TOE

The TOE is intended to be used as a component of the itsme mobile user authentication technology, to conduct the following high-level functions:

- generating a key pair and signature from a user's PIN (client subsystem);
- creating an authenticated set of messages (client subsystem);
- verifying an authenticated set of messages (server subsystem);
- creating second factor attestations (server subsystem); and
- generating audit records (server subsystem).

The itsme mobile user authentication technology consists of two parts: one running on the user's mobile device, and one running on a backend. The TOE consists of a client and a server subsystem. The TOE client subsystem is to be used by the part of the itsme authentication technology running on a user's mobile device: the itsme authentication client (IAC). The TOE server subsystem is to be used by a part of the itsme authentication technology running on a backend server: the itsme second factor server (SFS).

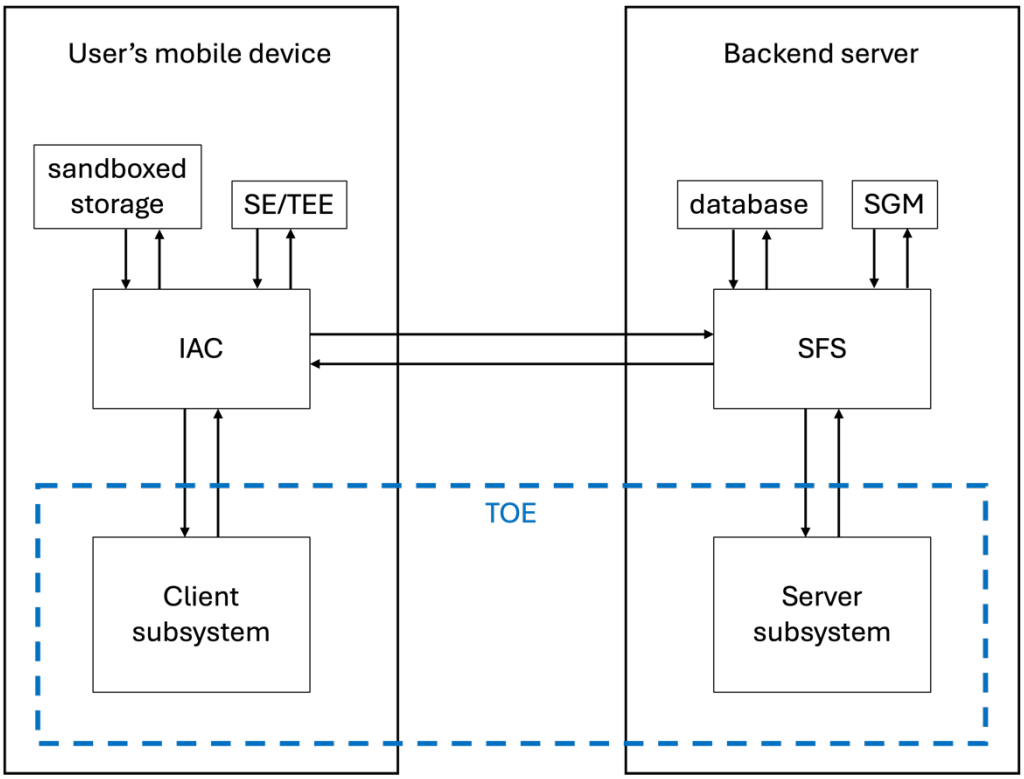

Figure 1 - TOE overview

The user interacts only with the TOE client subsystem via the IAC running on their mobile device. The IAC makes use of the application sandboxed storage (provided by the mobile OS) and interacts with the secure hardware (SE/TEE) of the mobile device. Users can **enrol** themselves with the backend server using their mobile device and their second factor (PIN and/or biometrics[1]) and **verify** themselves with the same mobile device and second factor afterwards. Next to this, users can securely update their second factors and manage public

[1] The TOE client subsystem does not handle biometrics directly, but instead delegates the responsibility to the application for obtaining a signature that is protected by the user's biometrics (coming from the mobile device's secure hardware).

keys as part of the enrol or verify functionality. The TOE client system outputs a set of authenticated messages meant for the TOE server subsystem.

The itsme authentication technology is responsible for setting up the communication between the user's mobile device and the backend server. At the backend server, the SFS interacts with the TOE server subsystem on behalf of the user, by passing the messages it received from the TOE client subsystem to the **enrol** and **verify** functions. The SFS makes use of a database and interacts with a Signature Generation Module (SGM). The SGM allows to abstract the attestation signature generation to a separate software or hardware component. The SFS outputs an attestation and authentication result for the IAC to act upon.

1.3.1.1 Generating a key pair and signature from the user's PIN (client subsystem)

The TOE client subsystem generates a key pair and signature from the user's *PIN* and the stored *PINSecret*. During *PINPublicKey* registration the TOE client subsystem will generate a fresh *PINSecret*, which it will then use to generate the key pair and create a signature. During *PINPublicKey* verification, the TOE client subsystem will ensure that the same key pair is generated (if the user entered the correct *PIN*) to create the signature. After each operation, the key pair is erased, and the user's *PIN* is again needed to generate the key pair and signature.

1.3.1.2 Creating an authenticated set of messages (client subsystem)

The TOE client subsystem generates an authenticated set of messages (see Table 1). The set of messages always includes one of the main messages, i.e. EnrolMessage or VerifyMessage, depending on which client functionality is used. The set of messages, as a whole, is authenticated by cryptographically chaining the messages. This ensures that the entire set of messages comes from the user who supplied their second factor.

When the TOE client subsystem enrolment functionality is used, the TOE client subsystem constructs an EnrolMessage and at least one of RegisterPINMessage and RegisterBiometricMessage. Additional messages can be included (e.g. for SubjectPublicKey management), based on the input provided to the TOE client subsystem.

When the TOE client subsystem verify functionality is used, the TOE client subsystem constructs a VerifyMessage and one of VerifyPINMessage and VerifyBiometricMessage. Additional messages can be included (e.g. for second factor management or SubjectPublicKey management), based on the input provided to the TOE client subsystem.

Some of the messages contain a signature to authenticate that message's content. The TOE client subsystem creates signatures itself (*PINSignature*), or can request a signature from the environment (*BiometricSignature*, *ClientSignature*, *SubjectSignature*).

Signatures can include data to be signed (*DTBS*) and *SessionData* (e.g. timestamps, sequence numbers) from the environment. By carefully setting these inputs, the application that makes use of the TOE, can prevent replay attacks of a given set of authenticated messages. *DTBS* and *SessionData* serve a different purpose: *DTBS* is intended for the third party consuming the Attestation, while *SessionData* is meant for the consumer application that integrates the TOE to securely bind with the communication channel it sets up.

Table 1 - Messages

Main messages either *EnrolMessage* or *VerifyMessage* shall be included in every set of messages	
EnrolMessage	Message from the TOE client subsystem to enrol the user's *ClientPublicKey* at the TOE server subsystem. This message holds *DTBS*. This message is signed with the *ClientPrivateKey*. This message shall be supplemented with at least one of *RegisterBiometricMessage* and *RegisterPINMessage*.
VerifyMessage	Message from the TOE client subsystem to verify the user's *ClientSignature* at the TOE server subsystem. This message is signed with the *ClientPrivateKey*. This message shall be supplemented with either *VerifyPINMessage* or *VerifyBiometricMessage*.
VerifyPINMessage	Message, supplementary to *VerifyMessage*, from the TOE client subsystem to verify the user's *PINSignature* at the TOE server subsystem. This message holds *DTBS*. This message is signed with the *PINPrivateKey*.
VerifyBiometricMessage	Message, supplementary to *VerifyMessage*, from the TOE client subsystem to verify the user's *BiometricSignature* at the TOE server subsystem. This message holds DTBS. This message is signed with the *BiometricPrivateKey*.
Second factor management optional, at least one of *RegisterBiometricMessage* and *RegisterPINMessage* to supplement *EnrolMessage*	
RegisterBiometricMessage	Message from the TOE client subsystem to register the user's *BiometricPublicKey* at the TOE server subsystem. This message is signed with the *BiometricPrivateKey*.
RegisterPINMessage	Message from the TOE client subsystem to register the user's *PINPublicKey* at the TOE server subsystem. This message is signed with the *PINPrivateKey*.
RemoveBiometricMessage	Message from the TOE client subsystem to remove the user's *BiometricPublicKey* from the TOE server subsystem.
SubjectPublicKey management optional	
AddSubjectPublicKeyMessage	Message from the TOE client subsystem to add a user's *SubjectPublicKey* at the TOE server subsystem. This message is signed with the corresponding *SubjectPrivateKey*.

RemoveSubjectPublicKeyMessage	Message from the TOE client subsystem to remove a user's *SubjectPublicKey* from the TOE server subsystem.

1.3.1.3 Verifying an authenticated set of messages (server subsystem)

The TOE server subsystem verifies an authenticated set of messages that contains one of the main messages, i.e. *EnrolMessage* or *VerifyMessage*. This ensures that no message can be altered, added and/or removed from a valid output of a TOE client subsystem. The environment needs to provide the same *SessionData* to the TOE server subsystem for this authenticated set of messages to verify. By verifying different signatures on individual messages (after verifying the set of messages), the TOE server subsystem establishes:

- user authentication (*BiometricSignature*, *ClientSignature*, *PINSignature*); or
- that one has access to the corresponding private key (*BiometricPrivateKey*, *PINPrivateKey*, *SubjectPrivateKey*), e.g. when updating the user's *AuthenticationData* (*BiometricPublicKey*, *PINPublicKey* and set of *SubjectPublicKey*s).

1.3.1.4 Creating second factor attestations (server subsystem)

The TOE server subsystem, after establishing user authentication, generates a second factor attestation. A second factor attestation attests that:

- at a given time,
- for given data to be signed (*DTBS*),
- a given user authenticated themself by providing a correct second factor to a TOE client subsystem instance that has access to the same environment as which the user enrolled with,
- specifying which second factor was used, and
- that this user has registered a set of public keys (*SubjectPublicKey*s) for which they had access to the corresponding private keys at the time or registration.

1.3.1.5 Generating audit records (server subsystem)

The TOE server subsystem generates audit records for the following events:

- enrolment of a user,
- verification of a user,
- update of the user's *AuthenticationData*, and
- generation of an *Attestation*.

1.3.2 TOE type

The TOE is a software library, allowing a user to generate an attestation that their second factor was verified. The second factor attestation uniquely ties together the user's subject public key(s), the data to be signed and the fact that their second factor has been verified at a given time. The TOE also allows the user to securely update their second factors and/or subject public keys.

The TOE is intended to be built into a mobile application running on the user's mobile device (client subsystem) and into a backend server application (server subsystem). The TOE provides the core security functionality needed and leaves flexibility to the applications that are built around it.

1.3.3 Required non-TOE hardware/software/firmware

The TOE client subsystem requires following hardware and software to run:

- A mobile device which is running a mobile operating system (e.g. Android or iOS).
- A mobile application, running on the mobile operating system, that provides a user interface and interacts with the itsme authentication client. The mobile application embeds the itsme authentication client.
- The itsme authentication client (IAC) interacts with the TOE, the mobile device's secure element or trusted execution environment and the mobile app's sandboxed storage. The itsme authentication client embeds the TOE.
- The mobile device's secure element or trusted execution environment (SE/TEE), including for the generation of the biometric protected key pairs and signatures.
- The mobile application's sandboxed storage.

Note that the TOE inherently does not impose requirements with respect to the mobile operating system as it is not (directly) dependent on it. The IAC provides the interactions with the underlying mobile operating system and passes the necessary information to the TOE through the input and callbacks. The IAC can impose requirements on the mobile operating systems (e.g. minimal versions), but this is outside the scope of this ST. The TOE client subsystem software package (see section 1.4.2) is compiled to WebAssembly, which is designed to work independently of the platform. It is up to the IAC to provide a WebAssembly runtime, suited for the underlying platform.

The TOE server subsystem requires following hardware and software to run:

- Server hardware (amd64 architecture) and operating system (Linux).
- The itsme second factor server (SFS), running on the server's operating system, that interacts with the TOE, a signature generation module and a database. The itsme second factor server embeds the TOE.
- A signature generation module (SGM).
- A database.

Note that the TOE inherently does not impose requirements on the server hardware, server operating system or database as it is not (directly) dependent on these. The SFS that provides the interactions with these and passes the necessary information to the TOE through the input and callbacks. The SFS can impose requirements on these, but this is outside the scope of this ST. The TOE server subsystem software package (see section 1.4.2) is compiled to a shared library for Linux as the target operating system and amd64 as the target architecture.

1.4 TOE description

1.4.1 Wider application context

The itsme mobile user authentication technology has been developed to provide a highly secure user authentication, where the risk and responsibility to secure the authentication is no longer placed in a single system but shared over multiple systems.

The itsme mobile user authentication technology consists of the itsme authentication client, embedded in a mobile application, running in a relatively secure mobile device that is under the user's sole control. It is combined with the itsme authentication backend services, running on highly secure, access-protected servers.

The itsme Authentication Client (IAC) and itsme Second Factor Server (SFS) both incorporate the TOE. The itsme authentication client embeds the TOE and makes use of its client subsystem functionality to get a set of messages. The itsme second factor server embeds the TOE and makes use of its server subsystem functionality to get an authentication result and/or attestation.

The itsme authentication client (IAC) embeds the TOE client subsystem.

- The IAC, depending on whether the user has been enrolled yet, makes use of the TOE client subsystem Enrol or Verify functions.
- The TOE client subsystem returns a set of authenticated messages for the IAC to pass along to the SFS.

The itsme second factor server (SFS) embeds the TOE server subsystem to return an Attestation to the IAC.

- The SFS, upon receiving a set of messages from the IAC, looks for an *EnrolMessage* or *VerifyMessage* within this set.
    a. For an *EnrolMessage*, it calls the TOE server subsystem enrol function.
    b. From the *VerifyMessage*, it deduces the user's claimed identity. If allowed by its internal policy for this user, it calls the TOE server subsystem verify function.
- The TOE server subsystem returns a combination of the following (where applicable):
    - the *AuthenticationResult*,
    - an *Attestation*,
    - the (updated) user's *AuthenticationData, and*
    - one or more *AuditRecord*(s).
- The SFS combines the *AuthenticationResult*, and the set of integrity validated messages with its internal policy to determine the result to the request of the IAC. Depending on the result, it updates the user's *AuthenticationData* in its database, handles the PIN verification outcome, and/or securely stores the *AuditRecord*(s). It communicates this result back to the IAC, and if applicable, the Attestation.

The itsme authentication client (IAC) acts upon the messages received from the SFS.

- The IAC generates one or more *SubjectSignature*(s) and unseals the attestation for the used *SubjectPublicKey*(s).

The combination of the *SubjectSignature*(s) together with the "unmasked" *Attestation* provides strong proof that the user has used this IAC instance and has verified their second factor at a given time. This can be used, for example, as a strong activation for generating a remote qualified signature.

1.4.1.1 Example use case

A user has accounts at multiple parties (e.g. a bank, an insurance company, a webshop…) and/or for multiple applications (e.g. logging in, approving transactions, signing documents). For each of these parties/applications, the user has a dedicated key pair, allowing the user to create signatures. These key pairs are called SubjectKeys, as they all link back to the same subject, the user.

Many of these parties/applications will require an authentication of the user itself, with the PIN of the user, not just a signature with the respective SubjectKey. Ideally the user should be able to use the same PIN code across parties/applications, without exposing this PIN code

to any of these parties/applications and whilst also limiting the number of attempts for the PIN code. These requirements rule out direct verification of the PIN code by each party/application separately. The TOE solves this problem, by centralizing PIN(/biometric) verification and delivering attestations showing the PIN was indeed correct and linking to the SubjectKeys of the user.

The process of authorising a bank transaction, could look like this:

1. The user first interacts with the bank, and obtains an identifier for the proposed transaction.
2. The user enters his PIN code to confirm the transaction.
3. The TOE client subsystem is called, with the PIN code entered by the user and the transaction identifier (and possibly other transaction details) as Data To Be Signed (DTBS). The TOE client subsystem produces a set of messages.
4. The user submits these messages to the TOE server subsystem, which will (if the PIN code is correct) return a signed attestation. The attestation will confirm:
    a. that the PIN was validated
    b. that the user registered a set of SubjectPublicKeys (among which the public key for banking transactions)
5. The user now submits the attestation, along with a signature put with the SubjectKey for banking transactions to the bank.
6. The bank verifies the signature with the SubjectPublicKey, checks that the attestation contains that SubjectPublicKey and validates the signature on the attestation (the bank holds a copy of the attestation public key of the TOE server subsystem).

1.4.2 Physical scope

The physical scope of the TOE consists of:

- itsme Second Factor Attestation Engine client subsystem software package, version 1.0.0, delivered as WebAssembly (Wasm).
- itsme Second Factor Attestation Engine server subsystem software package, version 1.0.0, delivered as a shared library.
- Guidance for TOE operators [Guidance], delivered in PDF format.

The TOE client subsystem is compiled to the WebAssembly binary instruction format, such that it can run cross-platform: on mobile operating systems like Android or iOS; and on different CPU architectures.

The TOE is delivered to the consumer of the TOE through a protected online repository, which requires the consumer to authenticate before granting access to the TOE.

1.4.3 Logical scope

1.4.3.1.1 TOE client subsystem Enrol and Verify functions

The Enrol and Verify functions take a combination of the following as input:

- a signature callback for the client private key (*ClientPrivateKey/C*),
- a *PIN* for registration
- a *PIN* for verification,
- the *PINSecret* for verification,
- a signature callback for the biometric private key (*BiometricPrivateKey/C*) for registration,

- a signature callback for the biometric private key (*BiometricPrivateKey/C*) for verification,
- a boolean indicating whether the *BiometricPublicKey* should be removed,
- one or more signature callback(s) for a subject private key (*SubjectPrivateKey/C*),
- one or more *SubjectPublicKey*(s) to be removed,
- the data to be signed (*DTBS*),
- the *SessionData*,
- a timestamp,
- a server instance identifier, and
- a seed for the random number generator.

On input of a user's *PIN* for registration, a new *PINKeyPair* and *PINSecret* is generated. On input of a user's *PIN* for verification, the *PINKeyPair* is derived (which will be the same as the original one when providing a correct PIN).

The signature callback(s) for a private key allow the TOE client subsystem to request the corresponding public key and signature on a given message from the environment.

The TOE client subsystem will construct the right set of messages, corresponding to the function used and input provided, and have these signed either by itself (for the *PINSignature*) or by the environment for private keys that are not under its control. This signed set of messages is such that they are all linked together and authenticated as coming from this user. The output of these functions is an authenticated set of messages.

Table 2 – TOE client subsystem environment

Name	Description
BiometricPrivateKey	The *BiometricPrivateKey* corresponds to the *BiometricPublicKey*. This private key is generated and stored inside the SE/TEE of the mobile device, it never leaves the mobile device's SE/TEE. This private key can only be used after the user unlocks it by presenting their biometrics.
BiometricPublicKey	The *BiometricPublicKey* is the corresponding public key of the *BiometricPrivateKey*.
BiometricSignature	A signature that is generated using the *BiometricPrivateKey* inside the mobile device's SE/TEE on a message provided by the TOE client subsystem. Generating the required randomness for signature generation is part of the signature generation process and thus also handled by the mobile device's SE/TEE.
ClientPrivateKey	The *ClientPrivateKey* corresponds to the *ClientPublicKey*. This private key is managed by the TOE client subsystem environment.
ClientPublicKey	The *ClientPublicKey* is the corresponding public key of the *ClientPrivateKey*. It serves as the user's identity within the TOE scope.
ClientSignature	A signature that was generated by the TOE client subsystem environment using the *ClientPrivateKey* on a message provided by the TOE client subsystem. Generating the required randomness for

Name	Description
	signature generation is part of the signature generation process and thus also handled by the TOE client subsystem environment.
SubjectPrivateKey	The *SubjectPrivateKey* corresponds to the *SubjectPublicKey*. This private key is managed by the TOE client subsystem environment.
SubjectPublicKey	The *SubjectPublicKey* is the corresponding public key of the *SubjectPrivateKey.* This key is related to the user and is meant for usage outside the TOE scope. By registering a *SubjectPublicKey* with the TOE, the user can show to a third party that the verification of their second factor is tied to the *SubjectPublicKey* they use to authenticate to that third party.
SubjectSignature	A signature that was generated by the TOE client subsystem environment using the *SubjectPrivateKey* on a message provided by the TOE client subsystem. Generating the required randomness for signature generation is part of the signature generation process and thus also handled by the TOE client subsystem environment.

1.4.3.1.2 TOE server subsystem enrol and verify functions

The enrol and verify functions take a combination of the following as input:

- an authenticated set of messages,
- a signature callback for the attestation key (*AttestationPrivateKey/C*),
- the claimed user's *AuthenticationData*. For a call to the enrol function, the TOE will deduce the claimed user's *AuthenticationData* from the set of messages. For a call to the verify function, the TOE environment supplies the claimed user's *AuthenticationData*,
- the *SessionData*,
- a timestamp (that was also communicated to the TOE client subsystem),
- the current timestamp,
- the server instance id, and
- a masking key (optional) for sealing the *SubjectPublicKey*s in the *Attestation.*

The TOE server subsystem will validate that the user has authenticated the set of messages.

Depending on the received messages, the TOE server subsystem will update the user's *AuthenticationData*.

The TOE server subsystem will generate an attestation and have it signed by the environment using the signature callback for the attestation private key.

The TOE server subsystem will generate audit records.

The output of these functions is a combination of the following (where applicable):

- the *AuthenticationResult*,
- an *Attestation*,
- the (updated) user's *AuthenticationData*, and
- one or more *AuditRecord*(s).

Table 3 – TOE server subsystem environment

Name	Description
AttestationPrivateKey	The *AttestationPrivateKey* is the corresponding public key of the *AttestationPublicKey*. This private key is managed by the TOE server subsystem environment.
AttestationPublicKey	The *AttestationPublicKey* is the corresponding public key of the *AttestationPrivateKey*.
AttestationSignature	A signature that was generated by the TOE server subsystem environment using the *AttestationPrivateKey* on a message provided by the TOE server subsystem. Generating the required randomness for signature generation is part of the signature generation process and thus also handled by the TOE server subsystem environment.
BiometricPublicKey	The *BiometricPublicKey* is the corresponding public key of the *BiometricPrivateKey*. It is part of the user's *AuthenticationData*. This key is used to verify *BiometricSignature*s.
ClientPublicKey	The *ClientPublicKey* is the corresponding public key of the *ClientPrivateKey*. It serves as the user's identity within the TOE scope. This key is used to verify *ClientSignature*s.
PINPublicKey	The *PINPublicKey* is the corresponding public key of the *PINPrivateKey*. It is part of the user's *AuthenticationData*. This key is used to verify *PINSignature*s.
SubjectPublicKey	The *SubjectPublicKey* is the corresponding public key of the *SubjectPrivateKey*. It is part of the user's *AuthenticationData*. This key is used to verify *SubjectSignature*s.

2 Conformance claims

2.1 CC conformance claims

This ST conforms to the requirements of Common Criteria version 2022, revision 1. It is CC Part 2 conformant and CC Part 3 conformant.

2.2 PP claim

This ST does not claim conformance to any PP.

2.3 Package claim

This ST conforms to assurance package EAL3 augmented by ADV_FSP.4, ADV_IMP.1, ADV_TDS.3, ALC_FLR.1, and ALC_TAT.1.

3 Security problem definition

3.1 Assets

Table 4 and Table 5 list the assets for the TOE client and server subsystems.

Table 4 – TOE client subsystem assets

Name	Description	Security	Storage location
Messages	The TOE client subsystem creates an authenticated set of messages.	integrity	
PIN	The *PIN* is known by the user and entered when needed. The *PIN* is never stored and erased from the TOE client subsystem when no longer needed. It is used to derive a *PINPrivateKey*.	confidentiality	
PINPrivateKey	The TOE client subsystem generates or derives the *PINPrivateKey* from the *PIN* and the *PINSecret*. The *PINPrivateKey* is never stored and erased when no longer needed. It is used to generate a *PINSignature*.	confidentiality, integrity	
PINPublicKey	The *PINPublicKey* is the corresponding public key of the *PINPrivateKey*. The *PINPublicKey* is generated or derived when needed and never stored in the TOE client subsystem environment.	confidentiality, integrity	
PINSecret	The *PINSecret* is stored by the environment of the TOE client subsystem and used to derive the *PINPrivateKey*. The TOE client subsystem generates a *PINSecret* when generating a new *PINKeyPair* (e.g. when registering or updating the user's *PIN*).	confidentiality, integrity	In environment of TOE client subsystem.
PINSignature	The TOE client subsystem generates a signature with the PINPrivateKey. The PINSignature is never stored in the TOE client subsystem environment.	confidentiality, integrity	

Table 5 – TOE server subsystem assets

Name	Description	Security	Storage location
Attestation	The TOE server subsystem creates an attestation for an authenticated user (see section 1.3.1.4 for a more detailed description). The attestation is signed by an *AttestationSignature* (see Table 3) by the TOE server subsystem environment.	integrity	
AuditRecord	The TOE server subsystem creates an *AuditRecord* for each important system event. The TOE server subsystem environment needs to store these.	confidentiality, integrity	In environment of TOE server subsystem.
AuthenticationData	The user's authentication data is stored in the TOE server subsystem environment and is used by the TOE to authenticate the user. *AuthenticationData* consists of a	confidentiality, integrity	In environment of TOE server subsystem.

Name	Description	Security	Storage location
	BiometricPublicKey, *PINPublicKey* and set of *SubjectPublicKey*s that is associated with a given user (*ClientPublicKey*).		
ClientPublicKey	The *ClientPublicKey* serves as the user's identity and is stored in the environment of the server subsystem together with that user's *AuthenticationData*.	integrity	In environment of TOE server subsystem.

3.2 Threat agents

This TOE considers three types of threat agents: attackers, unauthenticated users, and authenticated users. All threat agents are assumed to have complete knowledge of the TOE.

- The attacker is located outside of the TOE. The attacker has high attack potential: unless otherwise specified, it is assumed that the attacker has full access to the operational environment in which the TOE is running.
- The user can only interact with the TOE through the interfaces exposed by the IAC and SFS. Users have no access to the sandboxed memory or storage of the IAC or keys stored in the SE/TEE for the IAC. Users have no access the environment of the TOE server subsystem, i.e. no access to the memory, storage, database or SGM. In its interactions with the TOE,
    - an unauthenticated user has not yet authenticated themself to the TOE;
    - an authenticated user has.

3.3 Threats

T.ATTESTATION_EXISTENTIAL_FORGERY
The attacker, without access to the *AttestationPrivateKey*, generates a fresh (for which they did not observe a valid attestation) valid *Attestation* for any data.

T.ATTESTATION_SELECTIVE_FORGERY
An authenticated user generates a fresh valid *Attestation* for *SubjectPublicKey*(s) that are not associated with the authenticated user.

T.ATTESTATION_UNAUTHENTICATED
An unauthenticated user obtains an *Attestation* from the TOE.

T.AUDIT_ALTERATION
An attacker changes the audit log data, possibly covering up their tracks.

T.AUTHENTICATION_DATA_CLONE
An authenticated user registers a *SubjectPublicKey* that has already been registered for another user.

T.AUTHENTICATION_DATA_UNAUTHENTICATED
An unauthenticated user registers (or removes) a second factor (i.e. *BiometricPublicKey* or *PINPublicKey*) or *SubjectPublicKey*(s). Alternatively, if an authenticated user succeeds to change the *AuthenticationData* of another user.

T.AUTHENTICATION_DATA_UNAUTHORISED
An authenticated user registers a second factor (i.e. *BiometricPublicKey* or *PINPublicKey*) or *SubjectPublicKey* without having access to the corresponding private key.

T.AUTHENTICATION_FORGERY
An attacker successfully impersonates an (already enrolled) user without having access to that user's *BiometricPrivateKey* or *PIN*.

T.PIN_BRUTE_FORCE_CLIENT
The attacker, with access to only the TOE client subsystem operational environments, brute forces the user's *PIN*.

T.PIN_BRUTE_FORCE_SERVER
The attacker, with access to only the TOE server subsystem operational environments, brute forces the user's *PIN*.

T.RANDOM
The attacker, with access to only the TOE server subsystem operational environment and able to guess random numbers, derives the value of the *PINSecret*, *PINPrivateKey* and/or the user's *PIN*.

T. SUBJECT_SIGNATURE_FORGERY
The attacker, without access to the *SubjectPrivateKey*, may generate a fresh (for which they did not observe a valid SubjectSignature) valid *SubjectSignature*.

3.4 Organisational security policies

P.PIN_STRENGTH
The app, incorporating the TOE client subsystem, shall enforce a PIN strength policy, making sure that the user's PIN is sufficiently long and complex.

P.RELIABLE_AUDIT
The TOE environment shall keep reliable audit records for TOE server subsystem events.

P.SUBJECT_PUBLIC_KEYS_UNIQUE
The server, incorporating the TOE server subsystem, shall only allow unique subject public keys to be registered.

P.UNSUCCESSFUL_AUTHENTICATION_EVENT
The server, incorporating the TOE server subsystem, shall have an unsuccessful authentication event policy, where users are blocked from trying to authenticate after a pre-defined number of consecutive unsuccessful authentication events.

P.USER_UNIQUE
The server, incorporating the TOE server subsystem, shall only allow a user to enrol once, i.e. only enrolments with a unique *ClientPublicKey* will be accepted.

3.5 Assumptions

A.BIOMETRIC_HARDWARE
It is assumed that the mobile operation system, on top of which the app runs, provides a secure, strong on-device biometric verification system, backed by an embedded secure element on the mobile device, that outputs a digital signature upon successful verification.

A.MOBILE_APPLICATION_SANDBOX
It is assumed that the mobile operating system, on top of which the app runs, provides isolation features for the memory, storage, and secure OS API calls (e.g. for interacting with the embedded secure element of a mobile device). The user runs a genuine version of the app that incorporates the TOE. An attacker, who potentially runs malicious apps on the user's mobile device, cannot access the RAM of the TOE, nor the persistent storage, nor access keys inside the embedded secure element associated with the app that incorporates the TOE.

A.SEED
It is assumed that the TOE client subsystem environment provides a seed for the TOE client subsystem random number generator of at least 1024 bit of length that has at least 256 bit min-entropy. Furthermore, it is assumed that the TOE client subsystem environment keeps this seed confidential and limits exposure of this seed to its relevant parts that interact with the TOE client.

A.SERVER_ACCESS_PROTECTED
It is assumed that access to the environment in which in the TOE server subsystem runs, is protected both physically and logically such that only authorized administrators get access. The administrator maintains a secure state for this environment and the TOE inside, including protection against unauthorised software and configuration changes. This TOE server subsystem environment is reasonably protected against denial-of-service attacks.

A.VIGILANT_USER
It is assumed that users follow best security practises (e.g. not root their devices, have a device lock mechanism in place, apply updates ensuring security patches) for their mobile devices and that they do not reveal confidential data, i.e. their *PIN*, to an attacker. Moreover, the user is also assumed to provide physical security for their devices and ensure that the device stays under their sole control. The TOE operates in a relatively secure environment, which is provided by the user.

4 Security objectives

4.1 Security objectives for the TOE

OT.Cryptography_Secure
The TOE uses well-known cryptographic algorithms to generate keypairs, using suitable randomness, and perform other cryptographic functions.

4.1.1 Client subsystem

OT.Messages_Authenticated
The TOE client subsystem shall ensure that messages to be communicated to the TOE server subsystem is authenticated.

OT.PIN_Offline_Brute_Force_Prevent_Client
The TOE client subsystem has no access to any reference point, i.e. its environment does not store *PINPublicKey* nor any prior *PINSignature* for a correct *PIN*. An attacker with access to the *PINSecret* is thus not able to launch an exhaustive offline brute force attack to try all possible *PIN*s.

4.1.2 Server subsystem

OT.Authentication_Data_Secure_Values
The TOE server subsystem shall ensure that the values for all incoming *AuthenticationData* (*PINPublicKey*, *BiometricPublicKey*, *SubjectPublicKey*) are valid public keys and that a proof is provided showing access to the corresponding private key.

OT.Audit_Events
The TOE server subsystem shall generate audit records about the important system events.

OT.PIN_Offline_Brute_Force_Prevent_Server
The TOE server subsystem has no access to any reference point, i.e. its environment does not store *PINSecret*. An attacker with access to the *PINPublicKey* is thus not able to launch an exhaustive offline brute force attack to try all possible *PIN*s.

OT.User_Authentication_Required
The TOE server subsystem shall protect the enrol and verify functions by validating that the user is authenticated.

4.2 Security objectives for the operational environment

4.2.1 Client subsystem

OE.Biometric_Hardware
The environment of the TOE client subsystem shall ensure that, if users want to authenticate with strong biometrics, that the on-device biometric sensor, backed by the mobile device's SE/TEE, is used.

OE.Communication_Confidential
The environment of the TOE client subsystem shall ensure that all communication to the environment of the server subsystem is confidential.

OE.Mobile_Application_Sandbox
The environment of the TOE client subsystem shall provide isolation features for the memory, persistent storage, and secure operating system API calls (including access to embedded secure elements) of mobile applications.

OE.PIN_Prevent_Capture
The environment of the TOE client subsystem shall implement measures to limit capturing the PIN through the user interface. Specifically screen capture should be prevented.

OE.PIN_Prevent_Guessing
The environment of the TOE client subsystem shall ensure that the user's *PIN* is sufficiently protected against guessing by requiring a certain complexity and length.

OE.Private_Key_Confidential
The environment of the TOE client subsystem shall keep private keys (*BiometricPrivateKey*, *ClientPrivateKey*, *SubjectPrivateKey*) confidential, i.e. only accessible to relevant parts of the environment and only used in line with the purpose these were created for (generating signatures).

OE.Seed_Secure
The environment of the TOE client subsystem shall provide a seed to the TOE client subsystem random number generator of at least 1024 bit of length that has at least 256 bit min-entropy. Furthermore, this seed shall be kept confidential, i.e. only accessible to relevant parts of the environment, not stored in permanent storage, not communicated and removed from the memory when no longer needed.

OE.Signature_Secure_Client
The environment of the TOE client subsystem shall generate electronic signatures (*BiometricSignature*, *ClientSignature*, *SubjectSignature*) that cannot be forged without knowledge of the corresponding private key (*BiometricPrivateKey*, *ClientPrivateKey*, *SubjectPrivateKey*).

OE.Storage_Confidential
The environment of the TOE client subsystem shall provide confidential storage for the *PINSecret*.

OE.Vigilant_User
The user must follow best security practices for their mobile devices (e.g. not root their devices, have a device lock mechanism in place, apply updates ensuring security patches) and not disclose private data (e.g. their PIN) to anybody else.

4.2.2 Server subsystem

OE.Authentication_Data_Securely_Managed
The environment of the TOE server subsystem shall manage the user's *AuthenticationData* in a secure manner.

OE.Audit_Log_Protected
The environment of the TOE server subsystem shall protect the integrity of the audit log and protect the audit log from unauthorised deletion.

OE.Private_Key_Confidential
The environment of the TOE server subsystem shall keep private keys (*AttestationPrivateKey*) confidential, i.e. only accessible to relevant parts of the environment and only used in line with the purpose these were created for (generating signatures).

OE.Server_Environment_Secure
The environment of the TOE server subsystem shall limit physical and logical access to authorised administrators. The TOE software, operational environment of the TOE server subsystem, and database shall be maintained by an authorised administrator in a secure state, including protection against unauthorised software and configuration changes.

OE.Signature_Secure_Server
The environment of the TOE server subsystem shall generate electronic signatures (*AttestationSignature*) that cannot be forged without knowledge of the private key (*AttestationPrivateKey*).

OE.Subject_Public_Key_Unique
The environment of the TOE server subsystem shall not allow the same *SubjectPublicKey* to be registered for different users.

OE.Timestamp_Trusted
The environment of the TOE server subsystem shall provide trusted timestamps.

OE.Unsuccessful_Authentication_Attempt_Policy
The environment of the TOE server subsystem shall implement an unsuccessful authentication attempt policy.

OE.User_Unique
The environment of the TOE server subsystem shall not allow a *ClientPublicKey* to be enrolled again.

4.3 Security objectives rationale

4.3.1 Mapping between the security problem definition and security objectives

Table 6 - Mapping between the SPD and TOE security objectives

	OT.Authentication_Data_Secure_Values	OT.Audit_Events	OT.Cryptography_Secure	OT.Messages_Authenticated	OT.PIN_Offline_Brute_Force_Prevent_Client	OT.PIN_Offline_Brute_Force_Prevent_Server	OT.User_Authentication_Required
T.ATTESTATION_EXISTENTIAL_FORGERY							
T.ATTESTATION_SELECTIVE_FORGERY							
T.ATTESTATION_UNAUTHENTICATED							X
T.AUDIT_ALTERATION							
T.AUTHENTICATION_DATA_CLONE							
T.AUTHENTICATION_DATA_UNAUTHENTICATED							X
T.AUTHENTICATION_DATA_UNAUTHORIZED	X		X	X			
T.AUTHENTICATION_FORGERY			X	X			
T.PIN_BRUTE_FORCE_CLIENT					X		
T.PIN_BRUTE_FORCE_SERVER						X	
T.RANDOM			X				
T.SUBJECT_SIGNATURE_FORGERY							
P.PIN_STRENGTH							
P.RELIABLE_AUDIT		X					
P.SUBJECT_PUBLIC_KEYS_UNIQUE							
P.UNSUCCESSFUL_AUTHENTICATION_EVENT							
P.USER_UNIQUE							

Table 7 - Mapping between the SPD and environment security objectives

	OE.Audit_Log_Protected	OE.Authentication_Data_Securely_Managed	OE.Biometric_Hardware	OE.Communication_Confidential	OE.Mobile_Application_Sandbox	OE.PIN_Prevent_Caputre	OE.PIN_Prevent_Guessing	OE.Private_Key_Confidential	OE.Seed_Secure	OE.Server_Environment_Secure	OE.Signature_Secure_Client	OE.Signature_Secure_Server	OE.Subject_Public_Key_Unique	OE.Storage_Confidential	OE.Timestamp_Trusted	OE.Unsuccessful_Authentication_Attempt_Policy	OE.User_Unqiue	OE.Vigilant_User
T.ATTESTATION_EXISTENTIAL_FORGERY												X						
T.ATTESTATION_SELECTIVE_FORGERY		X										X						
T.ATTESTATION_UNAUTHENTICATED												X						
T.AUDIT_ALTERATION	X														X			
T.AUTHENTICATION_DATA_CLONE											X		X					
T.AUTHENTICATION_DATA_UNAUTHENTICATED		X																
T.AUTHENTICATION_DATA_UNAUTHORIZED											X							
T.AUTHENTICATION_FORGERY		X				X					X					X		
T.PIN_BRUTE_FORCE_CLIENT		X		X			X											
T.PIN_BRUTE_FORCE_SERVER														X				
T.RANDOM									X									
T.SUBJECT_SIGNATURE_FORGERY											X							
P.PIN_STRENGTH							X											
P.RELIABLE_AUDIT	X														X			
P.SUBJECT_PUBLIC_KEYS_UNIQUE													X					
P.UNSUCCESSFUL_AUTHENTICATION_EVENT																X		
P.USER_UNIQUE																	X	
A.BIOMETRIC_HARDWARE			X					X										
A.MOBILE_APPLICATION_SANDBOX					X			X						X				
A.SEED									X									
A.SERVER_ACCESS_PROTECTED								X		X								
A.VIGILANT_USER																		X

4.3.2 Rationale for mitigating threats

T.ATTESTATION_EXISTENTIAL_FORGERY is mitigated by OE.Signature_Secure_Server, which ensures that operational environment of the TOE server subsystem generates an *AttestationSignature* that cannot be forged without access to the *AttestationPrivateKey*.

T.ATTESTATION_SELECTIVE_FORGERY is mitigated by:

- OE.Signature_Secure_Server, which ensures that operational environment of the TOE server subsystem generates an *AttestationSignature* that cannot be forged without access to the *AttestationPrivateKey*; and

- OE.Authentication_Data_Securely_Managed, which ensures that operational environment of the TOE server subsystem securely manages *SubjectPublicKey*(s) that relate to the authenticated user.

T.ATTESTATION_UNAUTHENTICATED is mitigated by:
- OE.Signature_Secure_Server, which ensures that operational environment of the TOE server subsystem generates an *AttestationSignature* that cannot be forged without access to the *AttestationPrivateKey*; and
- OT.User_Authentication_Required, which ensures that the server.enrol and server.verify functions validate that the user is authenticated. As such, no *Attestation* will be generated for an unauthenticated user.

T.AUDIT_ALTERATION is mitigated by:
- OE.Audit_Log_Protected, which ensures that the operational environment of the TOE server subsystem protects the audit records; and
- OE.Timestamp_Trusted, which ensures that the operational environment of the TOE server subsystem provides trusted timestamps.

T.AUTHENTICATION_DATA_CLONE is mitigated by:
- OE.Signature_Secure_Client, which ensures that *SubjectSignature* cannot be forged without having access to the *SubjectPrivateKey*; and
- OE.Subject_Public_Key_Unique, which ensures that the TOE environment does not accept a new *SubjectPublicKey* for a user if they already have that *SubjectPublicKey* in its database for another user.

T.AUTHENTICATION_DATA_UNAUTHENTICATED is mitigated by:
- OT.User_Authentication_Required, which ensures that the server.enrol and server.verify functions validate that the user is authenticated. As such, no unauthenticated user can register (or remove) a second factor or *SubjectPublicKey*(s); and
- OE.Authentication_Data_Securely_Managed, which ensures that an authenticated user cannot register (or remove) a second factor or *SubjectPublicKey*(s) for another user.

T.AUTHENTICATION_DATA_UNAUTHORIZED is mitigated by:
- OT.Cryptography_Secure, which ensures that *PINSignature* cannot be forged without having access to the *PINPrivateKey*;
- OE.Signature_Secure_Client, which ensures that *BiometricSignature* or *SubjectSignature* cannot be forged without having access to the *BiometricPrivateKey* or *SubjectPrivateKey*;
- OT.Messages_Authenticated, which ensures that a valid *PINSignature*, *BiometricSignature* or *SubjectSignature* cannot be lifted from previous messages; and
- OT.Authentication_Data_Secure_Values, which ensures that the TOE server subsystem validates each *PINSignature*, *BiometricSignature or SubjectSignature* before accepting the corresponding new value for *PINPublicKey*, *BiometricPublicKey* or *SubjectPublicKey*.

T.AUTHENTICATION_FORGERY is mitigated by:

- OT.Cryptography_Secure, which ensures that *PINSignature* cannot be forged without having access to the *PINPrivateKey*, and that PINPrivateKey cannot be generated without having access to the user's *PIN*;
- OE.PIN_Prevent_Capture, which ensures that the user interface does not leak the user's PIN.
- OE.Signature_Secure_Client, which ensures that *BiometricSignature* cannot be forged without having access to the *BiometricPrivateKey;*
- OT.Messages_Authenticated, which ensures that a valid *PINSignature*, *BiometricSignature* cannot be lifted from previous messages;
- OE.Authentication_Data_Securely_Managed, which ensures that the environment of the TOE server subsystem securely manages the user's *AuthenticationData*.

T.PIN_BRUTE_FORCE_CLIENT is mitigated by:

- OT.PIN_Offline_Brute_Force_Prevent_Client, which ensures that there is no reference point in the environment of the TOE client subsystem to be able to brute force the user's *PIN*;
- OE.PIN_Prevent_Guessing ensures that the environment of the TOE client subsystem enforces that user's *PIN* is of sufficient complexity and length;
- OE.Communication_Confidential, which ensures that previously transmitted *PINPublicKey* and *PINSignature*(s) remain confidential;
- OE.Authentication_Data_Securely_Managed ensures that the *PINPublicKey* is stored confidentially inside the operational environment of the TOE server subsystem; and
- OE.Unsuccessful_Authentication_Attempt_Policy, which ensures that the environment of the TOE server subsystem enforces an unsuccessful authentication attempt policy, preventing one from keeping on guessing the user's *PIN* after a number of unsuccessful attempts.

T.PIN_BRUTE_FORCE_SERVER is mitigated by:

- OT.PIN_Offline_Brute_Force_Prevent_Server, which ensures that there is no reference point in the environment of the TOE server subsystem to be able to brute force the user's *PIN*; and
- OE.Storage_Confidential ensures that the *PINSecret* is stored confidentially inside the operational environment of the TOE client subsystem.

T.RANDOM is mitigated by:

- OE.Seed_Secure, which provides a confidential seed with sufficient length and min-entropy to the TOE client subsystem; and
- OT.Cryptography_Secure ensures that the TOE client subsystem transforms this seed into randomness suitable for its key generation.

T.SUBJECT_SIGNATURE_FORGERY is mitigated by OE.Signature_Secure_Client, which ensures that operational environment of the TOE client subsystem generates a *SubjectSignature* that cannot be forged without access to the *SubjectPrivateKey*.

4.3.3 Rationale for fulfilling organisational policy requirements

P.PIN_STRENGTH is satisfied by OE.PIN_Prevent_Guessing, which provides that the user's *PIN* is of sufficient complexity and length.

P.RELIABLE_AUDIT is satisfied by:

- OT.Audit_Events, which provides audit records about important system events;
- OE.TimeStamp_Trusted, which provides reliable timestamps that are used in the audit records; and
- OE.Audit_Log_Protected, which provides that the audit log is integrity-protected and that individual entries are protected from unauthorized deletion.

P.SUBJECT_PUBLIC_KEY_UNIQUE is satisfied by OE.Subject_Public_Key_Unique, which provides that every subject public key is associated with at most one user.

P.UNSUCCESSFUL_AUTHENTICATION_EVENT is satisfied by OE.Unsuccessful_Authentication_Attempt_Policy, which provides that one cannot keep on guessing the user's PIN after a number of unsuccessful attempts.

P.USER_UNIQUE is satisfied by OE.User_Unique, which ensures that the environment of the TOE server subsystem does not enrol a user with an identical *ClientPublicKey* as to any of the previously enrolled users.

4.3.4 Rationale for fulfilling assumptions

A.BIOMETRIC_HARDWARE is satisfied by OE.Biometric_Hardware, which ensures that biometric verification, if allowed, is done with an on-device biometric sensor that is backed by the embedded secure element of that mobile device. Note that OE.Biometric_Hardware implies OE.Private_Key_Confidential for the *BiometricPrivateKey*.

A.MOBILE_APPLICATION_SANDBOX is satisfied by OE.Mobile_Application_Sandbox, which provides isolation and protects the storage and memory of the mobile application that embeds the TOE. Note that OE.Mobile_Application_Sandbox implies OE.Private_Key_Confidential for the *ClientPrivateKey*; and OE.Storage_Confidential.

A.SEED is satisfied by OE.Seed_Secure, which provides a confidential seed with the requested length and min-entropy.

A.SERVER_ACCESS_PROTECTED is satisfied by OE.Server_Environment_Secure, which ensures that the environment of the TOE server subsystem is protected and limits the exposure to physical attacks. Note that OE.Server_Environment_Secure implies OE.Private_Key_Confidential for the *AttestationPrivateKey*.

A.VIGILANT_USER is satisfied by OE.Vigilant_User, which ensures that the user follows best security practices.

5 Security requirements

5.1 Security functional requirements

The following convention is used for the SFRs below (Common Criteria version 2022, revision 1, Part 2):

- Iterations of the SFRs are denoted by a slash and the iteration indicator after the component, e.g. FCS_COP.1/ECDSA. An SFR component followed by a slash and an asterisk denotes all iterations of that component, e.g. FCS_COP.1/*.
- Assignments, refinements and selections are placed between square brackets. A superscript after the closing bracket indicates the operation, i.e. [...]^{a} for assignments, [...]^{r} for refinements, and [...]^{s} for selections.
- Strikethrough text indicates that text was removed from the original SFR. If this is the case, a note explains the reasoning behind.

5.1.1 Security Audit (FAU)

5.1.1.1 FAU_GEN.1

FAU_GEN.1.1
The TSF shall be able to generate audit data of the following auditable events:
~~a) Start-up and shutdown of the audit functions;~~
b) All auditable events for the [not specified]^{s} level of audit;
c) Other specifically defined auditable events:[

- Enrolment
- Verification
- *AuthenticationData* update
- *Attestation* generation]^{a}

Note that audit records are always generated during operation of the TOE server subsystem, and that there is no option to start or shutdown this generation.

FAU_GEN.1.2
The TSF shall record within the audit data at least the following information:
a) Date and time of the auditable event, type of event, subject identity (if applicable), and the outcome (success or failure) of the event;
b) For each auditable event type, based on the auditable event definitions of the functional components included in the PP, PP-Module, functional package or ST, [none]^{a}.

5.1.2 Cryptographic Support (FCS)

Note that the EdDSA and ECDSA algorithms are not defined by cryptographic key sizes but rather by the underlying elliptic curves. The security strength of an elliptic curve is directly related to the order of the basepoint. In general, if an elliptic curve has a basepoint of order n, then the security strength will be approximately one half of the bit length of n. Curve Ed25519 and Curve P-256 each provide approximately 128-bits of security [NIST SP 800-186].

5.1.2.1 Cryptographic Key Management (FCS_CKM)

5.1.2.1.1 FCS_CKM.1/EdDSA

FCS_CKM.1.1/EdDSA
The TSF shall generate ~~cryptographic keys~~ [*PINPrivateKey*, *PINPublicKey*][r] in accordance with a specified cryptographic key generation algorithm [

1. The *PINPrivateKey* is derived from a *PINSecret* that is generated at random and *PIN,* using an HKDF.
2. This *PINPrivateKey* is then used as input for the EdDSA Key Pair Generation.][a]

and specified cryptographic ~~key sizes~~ [curves][r] [Ed25519][a] that meet the following: [[RFC 5869], and [FIPS 186-5], Appendix A.2.3][a].

5.1.2.1.2 FCS_CKM.5/EdDSA

FCS_CKM.5.1/EdDSA
The TSF shall derive ~~cryptographic keys~~ [*PINPrivateKey*, *PINPublicKey*][r] from [*PIN*, *PINSecret*][a] in accordance with a specified cryptographic key generation algorithm [

1. The *PINPrivateKey* is derived from the *PINSecret* and *PIN,* using an HKDF.
2. This *PINPrivateKey* is then used as input for the EdDSA Key Pair Generation.][a]

and specified cryptographic ~~key sizes~~ [curves][r] [Ed25519][a] that meet the following: [[RFC 5869], and [FIPS 186-5], Appendix A.2.3][a].

5.1.2.1.3 FCS_CKM.6

FCS_CKM.6.1
The TSF shall destroy [*PINPrivateKey*, *PIN*][a] when [no longer needed][s].

FCS_CKM.6.2
The TSF shall destroy cryptographic keys and keying material specified by FCS_CKM.6.1 in accordance with a specified cryptographic key destruction method [overwrite by zeros][a] that meets the following: [none][a].

5.1.2.2 Cryptographic Operation (FCS_COP)

5.1.2.2.1 FCS_COP.1/ECDSA

FCS_COP.1.1/ECDSA
The TSF shall perform [signature verification][a] in accordance with a specified cryptographic algorithm [ECDSA][a] and cryptographic ~~key sizes~~ [curves][r] [P-256][a] that meet the following: [[FIPS 186-5], Section 6.4.2][a].

5.1.2.2.2 FCS_COP.1/EdDSA

FCS_COP.1.1/EdDSA
The TSF shall perform [signature generation and/or verification][a] in accordance with a specified cryptographic algorithm [EdDSA][a] and cryptographic ~~key sizes~~ [curves][r] [Ed25519][a] that meet the following: [[FIPS 186-5], Section 7.6 and 7.7][a].

5.1.2.2.3 FCS_COP.1/HMAC

FCS_COP.1.1/HMAC
The TSF shall perform [keyed-hash message authentication code computation][a] in accordance with a specified cryptographic algorithm [HMAC][a] and cryptographic key sizes [256 bits, 512 bits][a] that meet the following: [[FIPS 198-1], Section 4][a].

5.1.2.2.4 FCS_COP.1/SHA-256

FCS_COP.1.1/SHA-256
The TSF shall perform [secure hash computation][a] in accordance with a specified cryptographic algorithm [SHA-256][a] ~~and cryptographic key sizes~~ that meet the following: [[FIPS 180-4], Section 6.2][a].

5.1.2.2.5 FCS_COP.1/SHA-512

FCS_COP.1.1/SHA-512
The TSF shall perform [secure hash computation][a] in accordance with a specified cryptographic algorithm [SHA-512][a] ~~and cryptographic key sizes~~ that meet the following: [[FIPS 180-4], Section 6.4][a].

5.1.2.3 Generation of random numbers (FCS_RNG)

The TOE only makes use of random numbers inside the client subsystem during the generation of *PINKeyPair*.

5.1.2.3.1 FCS_RNG.1

FCS_RNG.1.1
The TSF shall provide a [deterministic][s] random number generator that implements: [HMAC_DRBG, as defined in [NIST SP 800-90A], Section 10.1.2][a].

Note that the TOE verifies the statistical properties of the input for the DRBG (i.e. the seed it receives from the environment) by running the Repetition Count Test and Adaptive Proportion Test from [NIST SP 800-90B], Section 4.4.

FCS_RNG.1.2
The TSF shall provide [octets of bits][s] that meet [statistical test suites as defined in [NIST SP 800-90B], Section 5][a].

Note that [NIST SP 800-90B], Section 3.1.5.1 states that “HMAC, as specified in FIPS 198, with any approved hash function specified in FIPS 180 or FIPS 202” is a vetted algorithm for a keyed conditioning component, and that “Vetted conditioning components are permitted to claim full entropy outputs”.

5.1.3 Identification and authentication (FIA)

5.1.3.1 User Authentication (FIA_UAU)

5.1.3.1.1 FIA_UAU.2

FIA_UAU.2.1
The TSF shall require each user to be successfully authenticated before allowing any other TSF-mediated actions on behalf of that user.

5.1.3.1.2 FIA_UAU.5

FIA_UAU.5.1
The TSF shall provide [a possession-based, a knowledge-based and an inherence-based authentication mechanism][a] to support user authentication.

FIA_UAU.5.2/Enrol
The TSF shall authenticate any user's claimed identity [, following a server.enrol request,][r] according to the [following input information:

1. A set of messages.

and algorithm:

1. The TOE validates that the set of messages contains:
   - exactly one *EnrolMessage*,
   - at most one *RegisterPINMessage*,
   - at most one *RegisterBiometricMessage*, and
   - at least one *RegisterPINMessage* or *RegisterBiometricMessage*.
2. The TOE extracts the claimed user's *AuthenticationData* from the *EnrolMessage*, *RegisterPINMessage* and/or *RegisterBiometricMessage*.
3. The TOE validates the *ClientSignature* (possession-based factor) from the *EnrolMessage*, using the *ClientPublicKey* from the claimed user's *AuthenticationData*.
4. If applicable, the TOE validates the *PINSignature* (knowledge-based factor) from the *RegisterPINMessage* using the *PINPublicKey* of the claimed user's *AuthenticationData*.
5. If applicable, the TOE validates the *BiometricSignature* (inherence-based factor) from the *RegisterBiometricMessage* using the *BiometricPublicKey* of the claimed user's *AuthenticationData*.

   If all validations are positive, the user has been authenticated. The TOE assigns the role R.Authenticated_User to the user.

][a].

FIA_UAU.5.2/Verify
The TSF shall authenticate any user's claimed identity [, following a server.verify request,][r] according to the [following input information:

1. A set of messages.
2. The claimed user's *AuthenticationData*.

and algorithm:

1. The TOE validates that the set of messages contains:
   - exactly one *VerifyMessage*, and
   - exactly one of *VerifyPINMessage* or *VerifyBiometricMessage*.
2. The TOE validates the *ClientSignature* (possession-based factor) from the VerifyMessage, using the *ClientPublicKey* from the claimed user's *AuthenticationData*.
3. If applicable, the TOE validates the *PINSignature* (knowledge-based factor) from the *VerifyPINMessage* using the *PINPublicKey* of the claimed user's *AuthenticationData*.
4. If applicable, the TOE validates the *BiometricSignature* (inherence-based factor) from the *VerifyBiometricMessage* using the *BiometricPublicKey* of the claimed user's *AuthenticationData*.

   If all validations are positive, the user has been authenticated. The TOE assigns the role R.Authenticated_User to the user.

][a].

5.1.3.2 User Identification (FIA_UID)

5.1.3.2.1 FIA_UID.2

FIA_UID.2.1
The TSF shall require each user to be successfully identified before allowing any TSF-mediated actions on behalf of that user.

5.1.4 Security management (FMT)

5.1.4.1 Management of TSF data (FMT_MTD)

5.1.4.1.1 FMT_MTD.1

FMT_MTD.1.1
The TSF shall restrict the ability to [modify][s] the [*BiometricPublicKey*, *PINPublicKey*, *SubjectPublicKey*][a] to [R.Authenticated_User][a].

5.1.4.1.2 FMT_MTD.3

FMT_MTD.3.1
The TSF shall ensure that only secure values are accepted for [*BiometricPublicKey*, *ClientPublicKey*, *PINPublicKey*, *SubjectPublicKey*][a].

Note that secure values for the above public keys means that these public keys are of the right format and that the corresponding signature in the *RegisterBiometricMessage*, *EnrolMessage*, *RegisterPINMessage*, or *AddSubjectPublicKeyMessage* verifies, i.e. that a proof in zero knowledge is provided that one had access to the corresponding private key.

5.1.4.2 Specification of Management Functions (FMT_SMF)

5.1.4.2.1 FMT_SMF.1

FMT_SMF.1.1
The TSF shall be capable of performing the following management functions: [server.enrol, server.verify][a].

5.1.4.3 Security Management Roles (FMT_SMR)

5.1.4.3.1 FMT_SMR.1

FMT_SMR.1.1
The TSF shall maintain the roles [R.Authenticated_User][a].

FMT_SMR.1.2
The TSF shall be able to associate users with roles.

5.1.5 Protection of the TSF (FPT)

5.1.5.1 Integrity of exported TSF data (FPT_ITI)

5.1.5.1.1 FPT_ITI.1

FPT_ITI.1.1
The TSF shall provide the capability to detect modification of all TSF data during transmission between the TSF and another trusted IT product within the following metric: [signature integrity protection][a].

FPT_ITI.1.2
The TSF shall provide the capability to verify the integrity of all TSF data transmitted between the TSF and another trusted IT product and perform [operation abortion][a] if modifications are detected.

5.1.6 SFR dependencies analysis

Table 8 - SFR dependencies analysis

SFR	Dependencies	Fulfilled by
FAU_GEN.1	FPT_STM.1	see application note 1
FCS_CKM.1/EdDSA	FCS_CKM.2, FCS_CKM.5, or FCS_COP.1	FCS_COP.1/EdDSA
	FCS_RBG.1, or FCS_RNG.1	FCS_RNG.1
	FCS_CKM.6	FCS_CKM.6
FCS_CKM.5/EdDSA	FCS_CKM.2, or FCS_COP.1	FCS_COP.1/EdDSA
	FCS_CKM.6	FCS_CKM.6
FCS_CKM.6	FDP_ITC.1, FDP_ITC.2, FCS_CKM.1, or FCS_CKM.5	FCS_CKM.1/EdDSA, or FCS_CKM.5/EdDSA
FCS_COP.1/ECDSA	FDP_ITC.1, FDP_ITC.2, FCS_CKM.1, or FCS_CKM.5	see application note 2
	FCS_CKM.6	see application note 3
FCS_COP.1/EdDSA	FDP_ITC.1, FDP_ITC.2, FCS_CKM.1, or FCS_CKM.5	FCS_CKM.1/EdDSA, or FCS_CKM.5/EdDSA
	FCS_CKM.6	FCS_CKM.6
FCS_COP.1/HMAC	FDP_ITC.1, FDP_ITC.2, FCS_CKM.1, or FCS_CKM.5	see application note 4
	FCS_CKM.6	see application note 5
FCS_COP.1/SHA-256	FDP_ITC.1, FDP_ITC.2, FCS_CKM.1, or FCS_CKM.5	see application note 6
	FCS_CKM.6	see application note 6
FCS_COP.1/SHA-512	FDP_ITC.1, FDP_ITC.2, FCS_CKM.1, or FCS_CKM.5	see application note 6
	FCS_CKM.6	see application note 6
FCS_RNG.1	none	
FIA_UAU.2	FIA_UID.1	FIA_UID.2 (hierarchical)
FIA_UAU.5	none	
FIA_UID.2	none	
FMT_MTD.1	FMT_SMR.1	FMT_SMR.1
	FMT_SMF.1	FMT_SMF.1
FMT_MTD.3	FMT_MTD.1	FMT_MTD.1
FMT_SMF.1	none	
FMT_SMR.1	FIA_UID.1	FIA_UID.2 (hierarchical)
FPT_ITI.1	none	

Application note 1

The FAU_GEN.1 dependency on FPT_STM.1 is not fulfilled, because the TSF relies on the operating system to provide trusted timestamps. The environment objective OE.Timestamp_Trusted is ensuring that the operating system is configured to synchronise the clock to the trusted time.

Application note 2

The FCS_COP.1/ECDSA dependency on FDP_ITC.1, FDP_ITC.2, FCS_CKM.1, or FCS_CKM.5 is not fulfilled, as it is only about public operations (i.e. operations that do not require a private or secret cryptographic key).

Application note 3

The public keys used for signature verification in FCS_COP.1/ECDSA are still needed to verify signatures later and hence shall not be destroyed.

Application note 4

The FCS_COP.1/HMAC dependency on FDP_ITC.1, FDP_ITC.2, FCS_CKM.1, or FCS_CKM.5 is not fulfilled as the keys for HMAC are either ephemeral or static.

Application note 5

The FCS_COP.1/HMAC dependency on, FCS_CKM.6 is not fulfilled as the keys for HMAC are either ephemeral or static.

Application note 6

With respect to FCS_COP.1/SHA-256 and FCS_COP.1/SHA-512, there is no key material, hence the dependencies do not apply.

5.2

The assurance level for this TOE is EAL3+, being EAL3 augmented with the Implementation representation of the TSF (ADV_IMP.1), Modular design (ADV_TDS.3), Well defined developer tools (ALC_TAT.1), Basic flaw remediation (ALC_FLR.1), and Complete functional specification (ADV_FSP.4) assurance components.

Evaluation assurance level 3 (EAL3) — Methodically tested and checked — permits a conscientious developer to gain maximum assurance from positive security engineering at the design stage without substantial alteration of existing sound development practices.

EAL3 provides assurance by a full ST and an analysis of the SFRs in that ST, using a functional and interface specification, guidance documentation and an architectural description of the design of the TOE, to understand the security behaviour.

The analysis is supported by independent testing of the TSF, evidence of developer testing based on the functional specification and TOE design, selective independent confirmation of the developer test results, and a vulnerability analysis (based on the functional specification, TOE design, security architecture description and guidance evidence provided) demonstrating resistance to penetration attackers with a basic attack potential.
EAL3 also provides assurance through the use of development environment controls, TOE configuration management and evidence of secure delivery procedures.

This EAL represents a meaningful increase in assurance from EAL2 by requiring more complete testing coverage of the security functionality and mechanisms and/or procedures that provide some confidence that the TOE will not be tampered with during development.

The implementation representation of the TOE is provided to the evaluator in a form that can be analysed by the evaluator. The implementation representation is used in analysis activities for other families (analysing the TOE design, for instance) to demonstrate that the TOE conforms its design and to provide a basis for analysis in other areas of the evaluation (e.g. the search for vulnerabilities). The implementation representation is in a form that captures the detailed internal workings of the TSF, e.g. software source code.

ADV_IMP.1 has dependencies on ADV_TDS.3 - Modular design and ALC_TAT.1 - Well defined developer tools. ADV_TDS.3 has a dependency on ADV_FSP.4 - Complete functional specification.

Flaw remediation requires that discovered security flaws be tracked and corrected by the developer. Although future compliance with flaw remediation procedures cannot be determined at the time of the TOE evaluation, it is possible to evaluate the policies and procedures that a developer has in place to track and correct flaws, and to distribute the flaw information and corrections.

ALC_FLR has no dependencies on other assurance components.

Table 9 - Security Assurance Components used in this ST

Assurance class	Assurance components
ADV: Development	ADV_ARC.1 Security architecture description **ADV_FSP.4 Complete functional specification** **ADV_IMP.1 Implementation representation of the TSF** **ADV_TDS.3 Modular design**
AGD: Guidance documents	AGD_OPE.1 Operational user guidance AGD_PRE.1 Preparative procedures
ALC: Life-cycle support	ALC_CMC.3 Authorization controls ALC_CMS.3 Implementation representation CM coverage ALC_DEL.1 Delivery procedures ALC_DVS.1 Identification of security measures **ALC_FLR.1 Basic flaw remediation** ALC_LCD.1 Developer defined life-cycle model **ALC_TAT.1 Well defined developer tools**
ASE: ST evaluation	ASE_CCL.1 Conformance claims ASE_ECD.1 Extended components definition ASE_INT.1 ST introduction ASE_OBJ.2 Security objectives ASE_REQ.2 Derived security requirements

Assurance class	Assurance components
	ASE_SPD.1 Security problem definition ASE_TSS.1 TOE summary specification
ATE: Tests	ATE_COV.2 Analysis of coverage ATE_DPT.1 Testing: basic design ATE_FUN.1 Functional testing ATE_IND.2 Independent testing – sample
AVA: Vulnerability assessment	AVA_VAN.2 Vulnerability analysis

5.3

Table 10 provides the mapping between the SFRs and the TOE security objectives.

Table 10 - Mapping between the SFRs and TOE security objectives

	OT.Authentication_Data_Secure_Values	OT.Audit_Events	OT.Cryptography_Secure	OT.Messages_Authenticated	OT.PIN_Offline_Brute_Force_Prevent_Client	OT.PIN_Offline_Brute_Force_Prevent_Server	OT.User_Authentication_Required
FAU_GEN.1		X					
FCS_CKM.1/EdDSA			X		X	X	
FCS_CKM.5/EdDSA			X		X	X	
FCS_CKM.6			X		X		
FCS_COP.1/ECDSA	X		X	X			X
FCS_COP.1/EdDSA	X		X	X	X	X	X
FCS_COP.1/HMAC	X		X	X	X	X	X
FCS_COP.1/|SHA-256			X				
FCS_COP.1/SHA-512	X		X	X	X	X	X
FCS_RNG.1			X		X	X	
FIA_UAU.2							X
FIA_UAU.5							X
FIA_UID.2							X
FMT_MTD.1	X						
FMT_MTD.3	X						
FMT_SMF.1	X						
FMT_SMR.1	X						
FPT_ITI.1				X			

OT.Authentication_Data_Secure_Values is provided by the following SFRs:

- FCS_COP.1/* ensures that well-known secure cryptographic algorithms are used.
- FMT_MTD.1 ensures that only users with role R.Authenticated_User can update the TSF data.
- FMT_MTD.3 ensures that any new TSF data is validated.

- FMT_SMF.1 ensures that the TOE can perform actions on behalf of a user.
- FMT_SMR.1 ensures that the TOE can assign roles to users.

OT.Audit_Events is provided by FAU_GEN.1 that ensures that audit data is generated for auditable events.

OT.Cryptography_Secure is provided by the following SFRs:
- FCS_CKM.1/EdDSA ensures that *PINPrivateKey* and *PINPublicKey* are generated using a secure algorithm.
- FCS_CKM.5/EdDSA ensures that *PINPrivateKey* and *PINPublicKey* are derived using a secure algorithm.
- FCS_CKM.6 ensures that *PINPrivateKey* and *PIN* are securely destroyed when no longer needed.
- FCS_COP.1/* ensures that well-known secure cryptographic algorithms are used.
- FCS_RNG.1 ensures that generated random numbers are of sufficient quality to be used for cryptographic purposes.

OT.Messages_Authenticated is provided by the following SFRs:
- FCS_COP.1/* ensures that well-known secure cryptographic algorithms are used.
- FPT_ITI.1 ensures the integrity of all transmitted TSF data.

OT.PIN_Offline_Brute_Force_Prevent_Client is provided by the following SFRs:
- FCS_CKM.1/EdDSA ensures that *PINPrivateKey* and *PINPublicKey* are generated using a secure algorithm.
- FCS_CKM.5/EdDSA ensures that *PINPrivateKey* and *PINPublicKey* are derived using a secure algorithm.
- FCS_CKM.6 ensures that *PINPrivateKey* and *PIN* are securely destroyed when no longer needed.
- FCS_COP.1/* ensures that well-known secure cryptographic algorithms are used.
- FCS_RNG.1 ensures that generated random numbers are of sufficient quality to be used for cryptographic purposes.

OT.PIN_Offline_Brute_Force_Prevent_Server is provided by the following SFRs:
- FCS_CKM.1/EdDSA ensures that *PINPrivateKey* and *PINPublicKey* are generated using a secure algorithm.
- FCS_CKM.5/EdDSA ensures that *PINPrivateKey* and *PINPublicKey* are derived using a secure algorithm.
- FCS_COP.1/* ensures that well-known secure cryptographic algorithms are used.
- FCS_RNG.1 ensures that generated random numbers are of sufficient quality to be used for cryptographic purposes.

OT.User_Authentication_Required is provided by the following SFRs:
- FCS_COP.1/* ensures that well-known secure cryptographic algorithms are used.
- FIA_UAU.2 ensures that TSF mediated actions can only be performed on behalf of authenticated users.
- FIA_UAU.5 ensures that the user is authenticated.
- FIA_UID.2 ensures that TSF mediated actions can only be performed on behalf of identified users.

6 TOE summary specification

This section describes how the TOE satisfies all the SFRs.

6.1 OE F

6.1.1 Attestation

The TOE server subsystem shall ensure that an *Attestation* is only generated (as part of the enrol or verify functions) for users with the role R.Authenticated_User (which is assigned upon verifying that the user is authenticated).

This security function implements FMT_SMF.1, and FMT_SMR.1.

6.1.2 AuditRecord

The TOE server subsystem generates audit records of the most important events: enrolling a new user, verifying an existing user, updating the user's *AuthenticationData*, generating an attestation.

This security function implements FAU_GEN.1.

6.1.3 AuthenticationData

The TOE server subsystem shall ensure that *AuthenticationData* can only be updated (as part of the enrol or verify functions) for users with the role R.Authenticated_User (which is assigned upon verifying that the user is authenticated), after validating that the new *AuthenticationData* are valid public keys and that the proof that shows access to the corresponding private key validates.

This security function implements FMT_MTD.1, FMT_MTD.3, FMT_SMF.1, and FMT_SMR.1.

6.1.4 CryptoAlgorithms

The TOE uses the following cryptographic algorithms:

- ECDSA on cryptographic curve P-256
- EdDSA on cryptographic curve Ed25519
- HMAC
- SHA-256
- SHA-512
- HMAC_DRBG

Applicable standards:

- NIST FIPS 186-5 - Digital Signature Standard
- NIST FIPS 198-1 - The Keyed-Hash Message Authentication Code (HMAC)
- NIST FIPS 180-4 - Secure Hash Standard
- NIST SP800-90A - Recommendation for Random Number Generation Using Deterministic Random Bit Generators

This security function implements FCS_CKM.1/EdDSA, FCS_CKM.5/EdDSA, FCS_COP.1/*, and FCS_RNG.1.

6.1.5 PINKeyPair

The TOE client subsystem generates or derives a key pair from the user's *PIN* and stored *PINSecret*. The *PIN* and *PINPrivateKey* are destroyed immediately after use (to generate a *PINSignature*).

This security function implements FCS_CKM.1/EdDSA, FCS_CKM.5/EdDSA, FCS_CKM.6, FCS_COP.1/EdDSA and FCS_RNG.1.

6.1.6 UserAuthentication

The TOE server subsystem performs multi-factor user authentication by validating signatures on the provided set of messages against the provided claimed user's *AuthenticationData*. By doing so, all messages within this set are also authenticated (and thus integrity protected).

This security function implements FIA_UAU.2, FIA_UAU.5, FIA_UID.2, and FPT_ITI.1.

6.2 M FR F

The table below shows the mapping between the SFRs and the TSFs, clearly showing that all SFRs are covered by the TSFs.

Table 11 - Mapping between the SFRs and TSFs

SFR	**TSF**
FAU_GEN.1	AuditRecord
FCS_CKM.1/EdDSA	CryptoAlgorithms PINKeyPair
FCS_CKM.5/EdDSA	CryptoAlgorithms PINKeyPair
FCS_CKM.6	PINKeyPair
FCS_COP.1/ECDSA	CryptoAlgorithms
FCS_COP.1/EdDSA	CryptoAlgorithms PINKeyPair
FCS_COP.1/HMAC	CryptoAlgorithms
FCS_COP.1/SHA-256	CryptoAlgorithms
FCS_COP.1/SHA-512	CryptoAlgorithms
FCS_RNG.1	CryptoAlgorithms PINKeyPair
FIA_UAU.2	UserAuthentication
FIA_UAU.5	UserAuthentication
FIA_UID.2.1	UserAuthentication
FMT_MTD.1	AuthenticationData
FMT_MTD.3	AuthenticationData
FMT_SMF.1	Attestation AuthenticationData
FMT_SMR.1	Attestation AuthenticationData
FPT_ITI.1	UserAuthentication

7 References

7.1 I references

[Guidance] itsme Second Factor Attestation Engine Operation Guide, version 1.2.5, 2025

7.2 External references

[FIPS 180-4] Federal Information Processing Standards Publication 180-4: Secure Hash Standard (SHS), National Institute of Standards and Technology, 2015, https://doi.org/10.6028/NIST.FIPS.180-4

[FIPS 186-5] Federal Information Processing Standards Publication 186-5: Digital Signature Standard (DSS), National Institute of Standards and Technology, 2023, https://doi.org/10.6028/NIST.FIPS.186-5

[FIPS 198-1] Federal Information Processing Standards Publication 198-1: The Keyed-Hash Message Authentication Code (HMAC), National Institute of Standards and Technology, 2008, https://doi.org/10.6028/NIST.FIPS.198-1

[NIST SP 800-90A] National Institute of Standards and Technology Special Publication 800-90A Revision 1: Recommendation for Random Number Generation Using Deterministic Random Bit Generators, Elaine Barker (NIST), John Kelsey (NIST), 2015, https://doi.org/10.6028/NIST.SP.800-90Ar1

[NIST SP 800-90B] National Institute of Standards and Technology Special Publication 800-90B: Recommendation for the Entropy Sources Used for Random Bit Generation, Meltem Sönmez Turan (NIST), Elaine Barker (NIST), John Kelsey (NIST), Kerry McKay (NIST), Mary Baish (NSA), Michael Boyle (NSA), 2018, https://doi.org/10.6028/NIST.SP.800-90B

[NIST SP 800-186] National Institute of Standards and Technology Special Publication 800-186: Recommendations for Discrete Logarithm-based Cryptography: Elliptic Curve Domain Parameters, Lily Chen (NIST), Dustin Moody (NIST), Andrew Regenscheid (NIST), Angela Robinson (NIST), Karen Randall (Randall Consulting), 2023, https://doi.org/10.6028/NIST.SP.800-186

[RFC 5869] Internet Engineering Task Force (IETF) Request for Comments 5869: HMAC-based Extract-and-Expand Key Derivation Function (HKDF), H. Krawczyk (IBM Reseach), P. Eronen (Nokia), 2010, https://datatracker.ietf.org/doc/html/rfc5869

Appendix

1 Glossary

AddSubjectPublicKeyMessage	See Table 1.
Attestation	See Table 5.
AttestationPrivateKey	See Table 3.
AttestationPrivateKey/C	Signature callback for the *AttestationPrivateKey*.
AttestationPublicKey	See Table 3.
AttestationSignature	See Table 3.
AuditRecord	See Table 5.
AuthenticationData	See Table 5.
AuthenticationResult	Result of user authentication from the TOE server subsystem.
BiometricPrivateKey	See Table 2.
BiometricPrivateKey/C	Signature callback for the *BiometricPrivateKey*.
BiometricPublicKey	See Table 2 and Table 3.
BiometricSignature	See Table 2.
ClientPrivateKey	See Table 2.
ClientPrivateKey/C	Signature callback for the *ClientPrivateKey*.
ClientPublicKey	See Table 2 and Table 3.
ClientSignature	See Table 2.
DTBS	Data to be signed.
EnrolMessage	See Table 1.
IAC	itsme authentication client.
itsme authentication technology	The itsme authentication technology includes the IAC and SFS.
PIN	See Table 4.
PINKeyPair	The *PINPrivateKey* and the *PINPublicKey*.

PINPrivateKey	See Table 4.
PINPublicKey	See Table 4 and Table 3.
PINSecret	See Table 4.
PINSignature	See Table 4.
RegisterBiometricMessage	See Table 1.
RegisterPINMessage	See Table 1.
RemoveBiometricMessage	See Table 1.
RemoveSubjectPublicKeyMessage	See Table 1.
SE/TEE	The mobile device's secure element (secure enclave) or trusted execution environment.
Second factor	The user's *PIN* or biometrics.
Secure hardware	See SE/TEE.
SessionData	Session data to be supplied by the environment (e.g. timestamps, sequence numbers).
SFS	Second factor server.
SGM	Signature generation module.
SubjectPrivateKey	See Table 2.
SubjectPrivateKey/C	Signature callback for the *SubjectPrivateKey*.
SubjectPublicKey	See Table 2 and Table 3.
SubjectSignature	See Table 2.
VerifyBiometricMessage	See Table 1.
VerifyMessage	See Table 1.
VerifyPINMessage	See Table 1.