Unique Identification Authority of India (UIDAI)

Bangla Sahib Road, Behind kali Mandir, Gole Market,

, New Delhi 110001

AADHAAR REGISTERED DEVICES

TECHNICAL SPECIFICATION - VERSION 2.0 (REVISION 6)

Oct 2019

1 Introduction

The Unique Identification Authority of India (UIDAI) has been created, with the mandate of providing a Unique Identity (Aadhaar) to all Indian residents. The UIDAI provides online authentication using demographic and biometric data.

1.1 Aadhaar Authentication at a Glance

Aadhaar authentication is the process wherein Aadhaar Number, along with other attributes, including biometrics, are submitted online to the Aadhaar system for its verification on the basis of information or data or documents available with it. During the authentication transaction, the resident's record is first selected using the Aadhaar Number and then the demographic/biometric inputs are matched against the stored data which was provided by the resident during enrolment/update process.

For latest documentation on Aadhaar authentication, see https://uidai.gov.in/ecosystem/authentication-devices-documents/authentication-documents.html

1.2 Target Audience and Prerequisites

This is a technical document and is targeted primarily at biometric device manufacturers/providers who want to build registered devices as per this specification for Aadhaar authentication ecosystem.

This document assumes that readers are fully familiar with Aadhaar authentication model, related terminology, and authentication API technology details. Before reading this document, readers must read the Aadhaar authentication API specification available at https://uidai.gov.in/images/resource/aadhaar_authentication_api_2_5.pdf

IMPORTANT NOTE: In this document, term "**Device Provider**" used to refer to a device manufacturer or any agency who has partnership with the manufacturer to manage device

certification and related software/security aspects of registered devices. Device provider should be an entity registered in India and is responsible for STQC certification, device key management (as per this spec), and any security or other responsibilities set forth by UIDAI as part of device provider ecosystem rules.

2 Registered Devices

This chapter describes the specification in detail for registered devices for biometric device providers and also provides details on registration flow before these can be used with larger host devices.

2.1 Public Devices

Before understanding registered devices and the need for it, it is important to understand how public devices work.

Public devices are biometric capture devices that provide Aadhaar compliant biometric data to the application, which, in turn encrypts the data before using for authentication purposes. Currently AUA/Sub-AUA applications manage the biometric capture feedback user experience, any validation, and encryption of PID block. With public devices, providers may or may not provide an easy to use libraries to application developers.

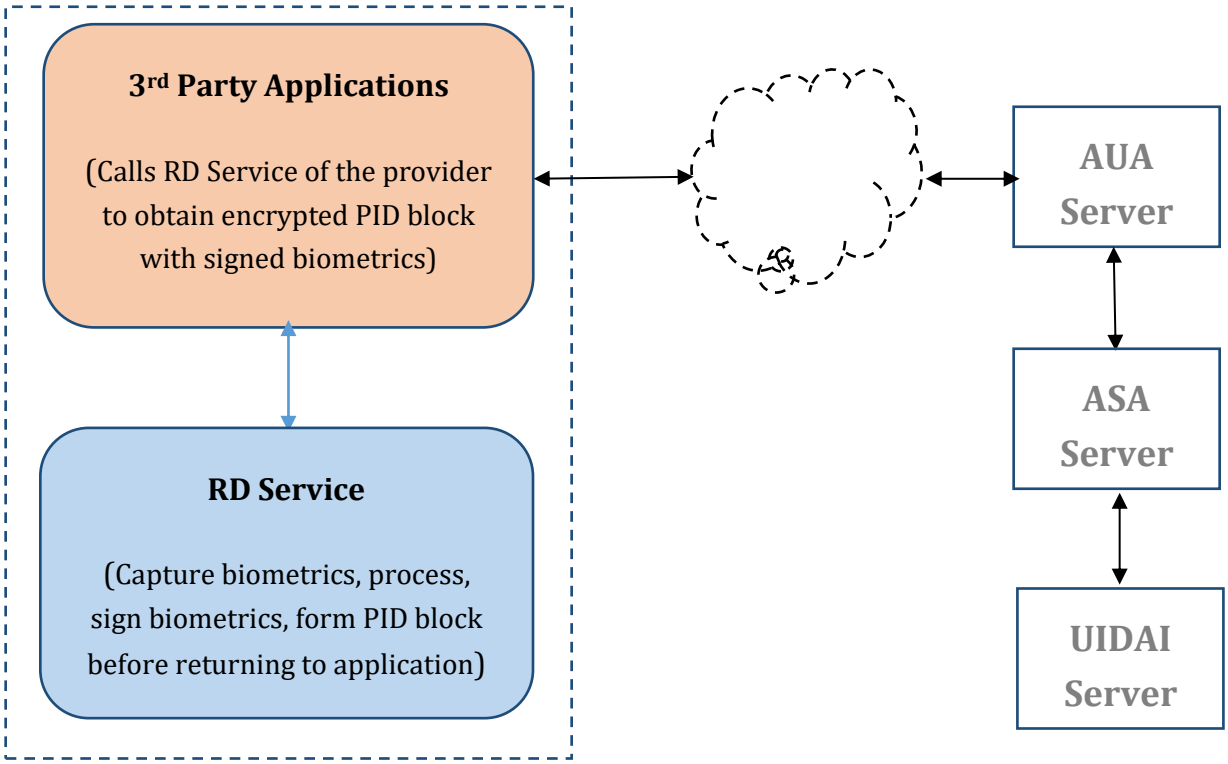
Several security measures are taken to ensure strong transaction security and end to end traceability even in public devices. These security measures fall into prevention and traceability. These include deploying signed applications, host and operator authentication by AUA, usage of multi-factor authentication, resident SMS/Email alerts on authentication, biometric locking, encryption/signing of sensitive data, and so on.

2.2 Registered Devices

Registered devices specification described in this document addresses the solution to eliminate the use of stored biometrics. It provides three key additional features compared to public devices:

1. **Device identification** – every device having a unique identifier allowing traceability, analytics, and fraud management.
2. **Eliminating use of stored biometrics** – biometric data is signed within the device using the device key to ensure it is indeed captured live. Then the Registered Device (RD) Service of the device provider must form the encrypted PID block before returning to the host application.
3. **A standardized RD Service provided by the device providers that is certified**. This RD Service (exposed via Service interface defined in this spec) encapsulates the biometric capture, any user experience while capture (such as preview), and signing and encryption of biometrics all within it.

Following is the logical diagram of a registered device (*for illustration purposes only, actual HW design may differ*). Detailed sequence diagrams are given later in the document.



There is no requirement for entire registered device to be physically separate unit. This is to ensure all devices (integrated and discrete) such as external devices connected to phones/laptops as well as biometric embedded phones, etc. can all act as registered devices.

NOTE: Rest of the document uses the term "RD Service" to refer to device provider's registered devices service that allows capture and processing of biometrics. This RD Service then returns encrypted PID block containing signed biometrics (using device private key within the registered devices secure zone) back to the calling application. **All registered devices providers MUST provide certified RD Service for various supporting operating systems so that applications can integrate easily in a secure and standard way without needing to embed any special software within applications**. Providers also must ensure that RD service can be run under separate user not needing root/admin privileges.

UIDAI **does not mandate any specific hardware design** and device providers are expected to innovate appropriate form factors for market use. Key design mandate is that registered devices **MUST securely sign the biometric data, form the encrypted PID block within the RD Service** and give it back to application for use within Aadhaar authentication.

Registered devices MUST ensure the following;

1. There should be no mechanism for any external program to provide stored biometrics and get it signed and encrypted.
2. There should be no mechanism for external program/probe to obtain device private key used for signing the biometrics.

It is important to note that it is in device provider's interest to ensure the above two items are implemented securely since any compromise on these will result in fraudulent activities signed using the device key. As per IT Act it is essential for the key owners (device provider) to protect the signature key and take responsibility for any compromise.

Following is the sequence of typical operations using the registered devices:

1. AUA/Sub-AUA provided application starts in host machine.
2. Application does a RD Service discovery (see later sections for details).
3. When ready for biometric input capture, application connects the RD Service to initiate the PID creation (which contains digitally signed and encrypted biometrics).
4. When the RD Service detects a good capture, it does necessary processing / extraction, creates the signed biometric record (FMR, FIR, IIR, FID), forms the encrypted PID block, and give the encrypted PID block back to application along with other details including Device Info.
5. Application obtains the encrypted PID block along with other information from the RD Service for calling Aadhaar authentication (see Aadhaar Authentication API 2.x specification for details).

2.3 Levels of Device Compliance

RD Service can be certified at 2 levels based on implementation.

2.3.1 Level 0 Compliance

Device security implementation has level 0 compliance if the signing and encryption of biometric is implemented within the software zone at host OS level. In this case, management of private keys need to be addressed carefully to ensure it is protected from access by users or external applications within the OS. All device providers should at a minimum obtain level 0 compliance and should not have mechanism to easily obtain the private key or inject biometrics. See later part of document for keystore implementation.

2.3.2 Level 1 Compliance

Device security implementation has Level 1 compliance if the signing and encryption of biometric is implemented within the Trusted Execution Environment (TEE) where host OS processes or host OS users do not have any mechanism to obtain the private key or inject biometrics. In this case, management of private keys need to be fully within the TEE. Any storage outside the TEE will require the keys to be wrapped using the TEE instance specific AES 256 bit keys.

The host OS should not have access to biometric capture except through the TEE. All of the processes related to create a biometric PID block must be executed within the TEE (at a level below the host OS) :

1. Biometric processing/extraction to create the bio element
2. Signing the bio element.
3. Encryption of the PID block

The following processes MUST take place within a hardware keystore (secure crypto block)

1. Identity of the chip Ci_k (look at section Pre-Certified Hardware Identity) should be stored in the secure crypto block or wrapped with the instance specific unique key (non extractable and stored within the secure crypto block) if stored outside. Chip identity should be non clonable.
2. Key pair generation
3. Signing the bio element

It is required to minimize the attack surface at the system level by using methods such as but not limited to hidden traces, protective meshing, encrypted communication etc. Minimizing the attack surface is inline with the objective 1 of this document.

In addition, it is recommended that tamper responsiveness be implemented for the system. UIDAI will differentiate between devices with and without tamper responsiveness.

2.4 Device Identity

One of the key aspects about Registered devices is to identify the devices uniquely. Between the L0 and L1 there are difference in identification of the devices.

L0 - In a L0 device we would use the following

- Idhash SHA-256 of any internal physical ID that is used to recognize physical device (such as serial number). This should be read automatically without any user input. This ID is not expected to change during the life of that physical device. **idHash MUST match what was sent during registration** (see Register API call later).

L1 - In a L1 device we would use the following

The CI_k key from the pre-certified hardware (Refer the section on Pre-Certified hardware later) is used to "sign" the device identity.

The device identity data would be computed as follows:

a. The device provider software should use the device serial number and timestamp as input (should match the timestamp value of PID) to construct the string TD.
   i. TD=deviceSerialNumber:base64(<deviceSerialNumber>);timestamp:<timestamp>
b. The device provider will construct the idHash in the following manner:
   i. Sign1 = DigSign(TD) (signed using the pre-certified hardware sign method)
   ii. idHash=**concat <deviceSerialNumber>** + _##_ + <base64(Sign1)>

Note: _##_ is a delimiter string.

Note: Max serial number would be 20 characters

The resultant idHash would be provided as idHash to the PID block for all L1 device.Unlike L0 the idHash here is expected to change for every auth call. .The management server should validate the signature in the idHash before registration and before key rotation. The idHash data has to be kept as a audit record for future verification and compliance needs. .The management server and the TEE software should have the ability to synchronize time a minimum once a day and host time should not be trusted. The deviceSerialNumber in the idHash computation should not change for lifetime while the signature depends on the timestamp.

2.5 PID Creation – Signing and Encrypting Biometrics

Providers of registered devices should:

1. Obtain the device provider ID from UIDAI.
2. Provide list of certified models (model code and other details).
3. Procure a digital certificate from a valid CA in India (refer 2.7 for supported algorithms) and get it signed by UIDAI. This would be the device provider key. Device providers can have one or more keys.
4. All devices should generate an asymmetric key pair within the device. This would be the device key. Every physical instance of the device should have its own device key.
5. Device public key should be signed by one of the device provider key. Refer section 2.7 for supported algorithms. Provider should sign the public key using the HSM server over a secure channel during key initialization.
6. Device provider MUST ensure each physical device has a unique code. To ensure device codes are globally unique it is necessary that device provider uses a 128-bit UUID Version 4 (represented in HEX notation).

Note: Device public-private key generation and signing of device public key with device provider key can be performed at any point of time during device's lifecycle. **However, the specific device key pair used to sign biometrics for the purposes of Aadhaar authentication should be used for UIDAI purposes only**.

Process of PID creation within RD Service is described below:

1. RD Service provides standard APIs as per UIDAI standards (see next section for details) to application developers to call whenever biometric capture is required.

2. RD Service should return service and device info (see later sections for API details) to calling applications which includes "device provider model ID (*Mi*)", "device public key cert (*Mc*) (signed by Device Provider Key)", and "Device Code (*Dc*)". These attributes are used by application to form device element of authentication XML (see Authentication API Specifications 2.x).
3. When RD Service is called, it should capture, process, sign the biometric record (FMR, FIR, IIR, FID) using device key, and form the encrypted PID block before returning the encrypted PID block with other informations to application.
4. Within PID block, every biometric record (bio element) should have "Biometric Signature (*Bs*)" for that data. A separate independent attribute is used instead of any signature that is embedded within bio record to ensure various image and ISO formats are supported.
5. RD Service should use the following logic to sign the biometric record:
    a. Bh = SHA-256(bio_record) of each successful biometric capture.
    b. Be = DSA(Bh+ts+Dc, Dpk) where;
        - *Dpk* is the device private key
        - *ts* is the PID timestamp (in String representation)
        - *Dc* is the unique device code in String format

    Refer section 2.7 for supported signature algorithms.
    c. Bs = base64(Be)
6. Within PID block, for the "*Bios"* element, attribute "*dih"* should be computed as:

    a. In case of L0 Devices:
        - dih= SHA-256(dpId+rdsId+rdsVer+dc+mi+idHash)
    b. For L1 Devices:
        - dih= Base64 {Base 64 [SHA-256(dpId+rdsId+rdsVer+dc+mi)]_&&_[idHash of L1 signed using the hardware signing method]}

    a. dpId – Device Provider ID as assigned during certification.
    b. rdsId – RD Service ID as assigned during certification.
    c. rdsVer – RD Service Version.
    d. mi – Device Provider Model ID.
    e. idHash – As defined in the device identity section earlier

7. Within PID block, wadh is added if passed by the calling application (see PidOptions→Opt element later).

8. After capturing the biometric, RD Service forms the encrypted PID block as per Aadhaar Authentication API 2.X specification and returns it to the application along with other information.
    a. Note that RD Service MUST store latest UIDAI public key used for PID encryption and should have a mechanism to initialize and update over the air.

2.6 RD Service APIs

All RD Services **MUST** provide the following standard APIs to ensure applications have a common way to interface with all Aadhaar compliant registered devices. **This is a necessary condition for certification of registered devices**.

2.6.1 Interface Methods

All RD Services must provide a standard interface having the following two methods and **MUST work without ANY state stored within driver across calls**.

```
// main capture call which takes PidOptions XML data as input and returns PidData XML as output
String capture (String pidOptions);
// utility method to obtain DeviceInfo XML from the RD Service
String getDeviceInfo ();
```

A mechanism to discover the RD Service and invoke these methods are described in later sections of this document.

2.6.2 Input/Output XMLs

```
<PidOptions ver="" env="">
  <Opts fCount="" fType="" iCount="" iType="" pCount="" pType="" format="" pidVer="" timeout="" otp="" wadh="" posh=""/>
  <Demo> Demographic Attributes as specified in authentication API </Demo>
  <CustOpts>
      <!-- no application should hard code these and should be configured on app or AUA servers. These parameters can be used for any custom application authentication or for other configuration parameters. Device providers can differentiate their service in the market by enabling advanced algorithms that applications can take advantage of. -->
    <Param name="" value="" />
  </CustOpts>
```

</PidOptions>

/*

PidOptions:

ver: (mandatory) Version under PidOptions is “1.0”. This is necessary to allow applications to gracefully upgrade the RD service. **RD Service must support current version and one previous version** to allow apps to upgrade at different points in time.

env: (optional) UIDAI Authentication environment for which capture is called. Valid values are "P" (Production), "PP" (Pre-Production), and "S" (Staging). If blank or if the attribute is not passed, RD service should default this to "P". This is provided to allow same RD service to use different UIDAI public key based on the environment.

Opts:

Int fCount (optional) number of finger records to be captured (0 to 10)

Int fType (optional) ISO format (0 for FMR or 1 for FIR), 0 (FMR) is default

Int iCount (optional) number of iris records to be captured (0 to 2)

Int iType (optional) ISO format (0 for IIR), 0 (IIR) is default

Int pCount (optional) number of face photo records to be captured (0 to 1). Currently face matching is not supported.

Int pType (optional) face format. Currently face matching is not supported.

Int format (mandatory) 0 for XML, 1 for Protobuf

String pidVer (mandatory) PID version

Int timeout (optional) capture timeout in milliseconds

String otp (optional) OTP value captured from user in case of 2-factor auth

String wadh (optional) If passed, RD Service should use this within PID block root element “as-is”.

String posh (optional) if specific positions need to be captured, applications can pass a comma delimited position attributes. See “posh” attribute definition in Authentication Specification for valid values. RD Service (if showing preview) can indicate the finger using this. If passed, this should be passed back within PID block. Default is “UNKNOWN”, meaning “any” finger/iris can be captured.

Demo:

Element allows demographic data to be passed to form PID block as per authentication specification

CustOpts:

Allows vendor specific options to be passed. Param element may repeat. Applications should avoid hard coding any custom parameters to ensure same application can work with multiple vendor devices.

*/

<PidData>

<Resp errCode="" errInfo="" fCount="" fType="" iCount="" iType="" pCount="" pType="" nmPoints="" qScore=""/>

<DeviceInfo > <DeviceInfo/>

<Skey ci="">encrypted and encoded session key</Skey>

<Hmac>SHA-256 Hash of Pid block, encrypted and then encoded</Hmac>

<Data type="X|P"> *base-64 encoded encrypted pid block* </Data>

</PidData>

/*

Resp:

Int errCode (mandatory) 0 if no error, else standard error codes

String errInfo (optional) additional info message in case of error/warning

Int fCount (mandatory for FP) number of finger records actually captured

Int fType (mandatory for FP) actual format type – 0 (FMR) or 1 (FIR)

Int iCount (mandatory for Iris) number of iris records actually captured

Int iType (mandatory for Iris) actual Iris format (0 for IIR)

Int pCount (mandatory for Photo) number of face photo records actually captured. Currently face matching is not supported.

Int pType (mandatory for Photo) face format. Currently face matching is not supported.

Int nmPoints (mandatory for FMR capture) Number of minutiae points when FMR is captured. Applications may use this for accepting or retrying the capture. If multiple fingers are captured, send comma delimited numbers.

Int qScore (optional) If quality check is done, send a normalized score that is between 0 and 100. Device providers may allow configuration within RD service to use specific quality check algorithms to be enabled. Either it can be configured within RD service or applications can pass those under PidOptions→CustOpts→Param.

Skey:

String skey (mandatory) encrypted session key as per auth spec

String ci (mandatory) UIDAI public key identifier as per auth spec

Hmac:

String hmac (mandatory) hmac value as per auth spec

*/

<DeviceInfo dpId="" rdsId="" rdsVer="" dc="" mi="" mc="" >

<Additional_Info>

```
    <!-- Info element may repeat -->
    <Info name="" value="" />
  </Additional_Info>
</DeviceInfo>
```

```
/*
DeviceInfo:
    dpId – (mandatory) Unique code assigned to registered device provider.
    rdsId – (mandatory) Unique ID of the certified registered device service.
    rdsVer – (mandatory) Registered devices service version.
    dc – (mandatory) Unique Registered device code.
    mi – (mandatory) Registered device model ID.
    mc – (mandatory) This attribute holds registered device public key certificate.
            This is signed with device provider key.
Additional_Info (optional):
        Optional element provided to pass back additional device info to application. Note that data passed
        through this MUST NOT violate UIDAI policies and device providers must take care in defining this to
        ensure no sensitive data is passed.

Info (optional): May repeat more than once.
    String name (mandatory) the name of the attribute
    String value (mandatory) value of the attribute
*/
```

Note: The entire <Demo> element should be inline with latest aadhaar authentication specification.

2.7 Registration and Key Management

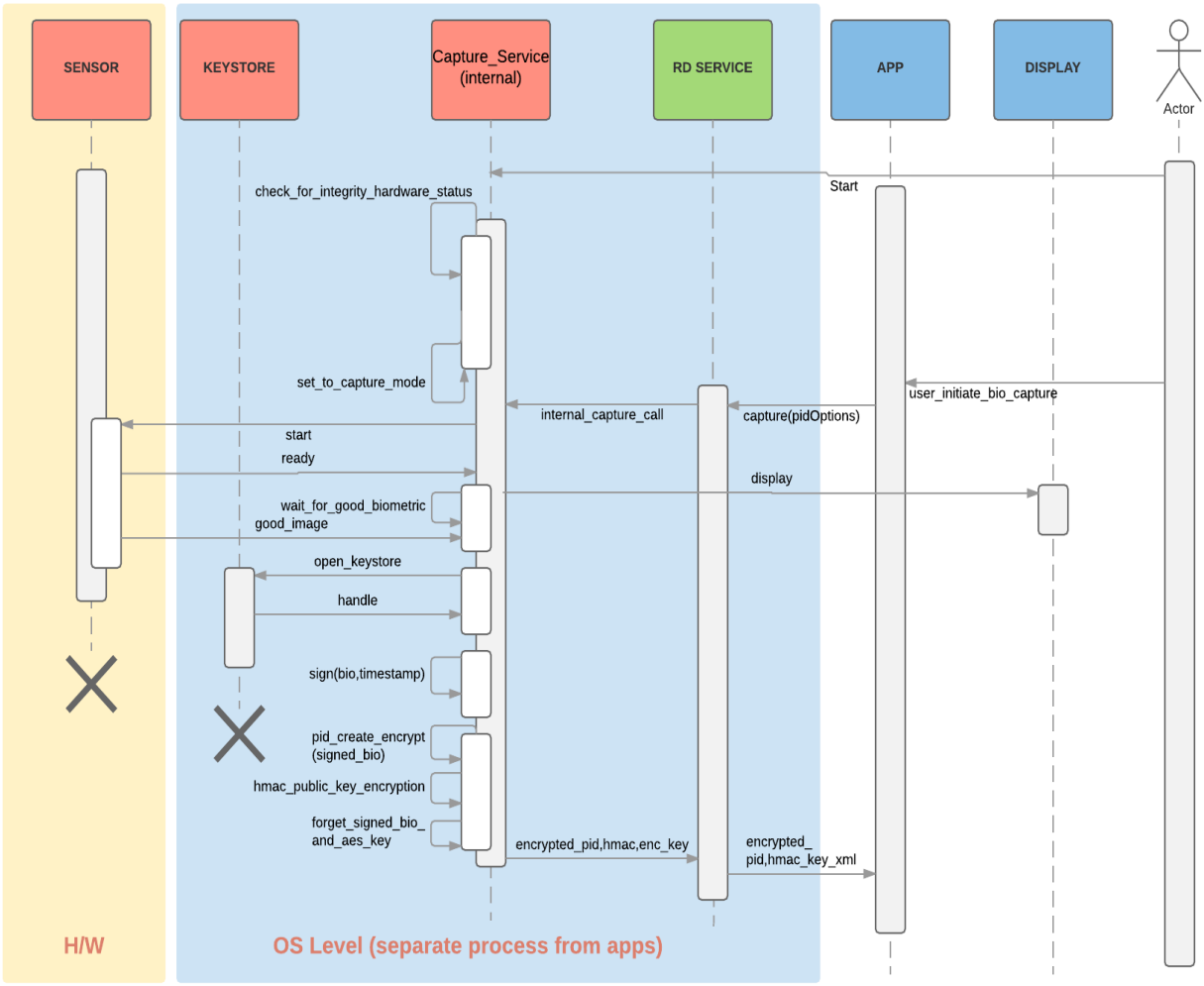
1. Device providers to register and obtain a device provider ID via UIDAI portal.
2. Device provider can register one or more public certificate procured from CA and get it signed by UIDAI. These are then used to sign the device public key certificate.
3. Device providers can rotate, revoke their keys via the UIDAI portal.

2.8 Certificates, Keys Policies

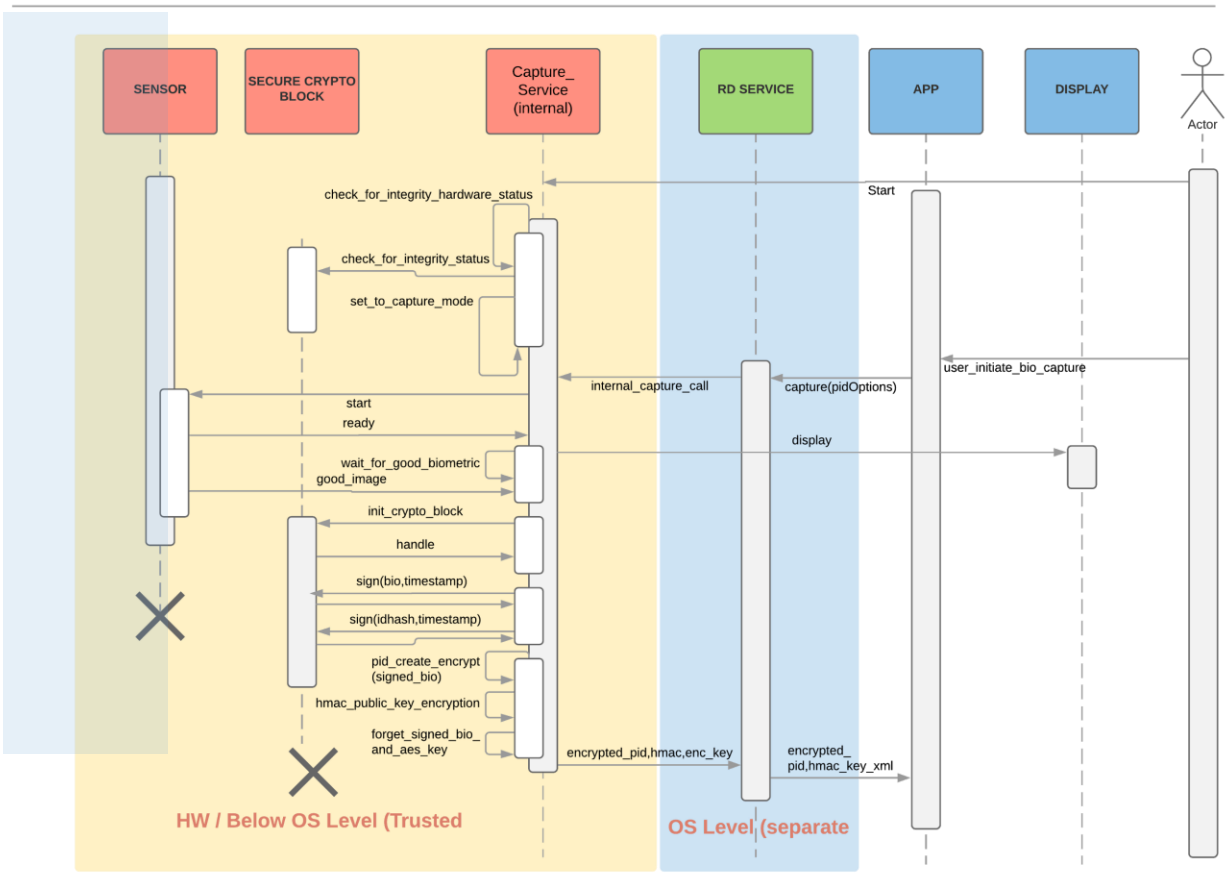
1. Below are the currently supported algorithms for digital signing.
    - SHA256withRSA (2048 bit key).
2. All device provider certificates should be procured from a certification authority (CA) as per Indian IT Act. (http://www.cca.gov.in/cca/?q=licensed_ca.html)
3. All device provider certificates should be class II or class III and X509 v3 compliant.
4. Organization attribute in the certificate's subject SHOULD match the device provider's name registered with UIDAI.
5. Device provider SHOULD have necessary server/backend infrastructure to sign the device public key, rotate device keys older than specified time as per UIDAI policy, update UIDAI public key used for encryption, and provide updates/fixes to their RD Service.

3 Sequence Diagrams

Sequence diagram for L0 device is given below:



Sequence diagram for L1 devices is given below:



4 "Certified RD Services" Registry

Certified RD services details are made available via a registry for applications to use. Applications are expected to check with this registry during RD service installation (if applications are managing these) and use.

Information about RD services in various status are made available at the following URL: https://authportal.uidai.gov.in/devices/rdservice_registry.xml/

- One provider has many models
- One provider has one or more RD service per OS (many versions)
- One RD service may handle many models

Following is the XML for the service registry.

```
<UIDAIRDServices lastUpdated="" ttl="">
  <!-- below element repeats for all distinct RD Services -->
  <RDService rdsId="" dpId="">
    <!-- below element repeats for each version that is certified -->
    <Version rdsVer="" rdsMD5="" osId="" osVer=""
status="ACTIVE|SUSPENDED|ONBOARDED|DEPRECATED" downloadUrl="" docUrl="">
      <!-- supported models within this version -->
      <ProviderModel mi="" type="F,I,P" level="L0|L1" />
    </Version>
  </RDService>
  </Signature>
</UIDAIRDServices>
```

/**

lastUpdated: Timestamp in YYYYMMDDhhmmss format depicting when this file was last updated.

ttl: Time to Live in hrs. This is an indicator for applications caching this file to refresh the cache with new file. After ttl time window has passed, applications should do HTTP HEAD request to check if the file has changed and download if changed.

RDService→rdsId: Unique alpha-numeric ID assigned to an RD Service post certification (separate per OS).

RDService→dpId: Unique Device Provider ID. Alpha-numeric.

RDService→Version: This element repeats to capture all certified versions across operating systems from this provider.

Version→rdsVer: Version of the RD Service.

Version→rdsMD5: MD5 checksum of the certified RD Service installable. Agencies installing RD Services should download and verify the checksum to ensure they are indeed using certified versions.

Version→osId: Enum depicting ID of the operating system for which this RD Service version is certified. Values can be LINUX, WINDOWS, ANDROID, WINDOWS MOBILE, etc.

Version→osVer: Version of the OS for which this RD Service is released. This can be a comma delimited string containing version numbers in the form “x.y.z”. Also, “+” sign may be there (e.g., 3.4+) depicting any version above can be used.

Version→status: Current status of the RD service version. Valid values are “ACTIVE” (for active ones), “SUSPENDED” (ones temporarily suspended due to non-compliance), “ONBOARDED” (ones being certified currently), “DEPRECATED” (ones that are deprecated from future usage).

Version→downloadUrl (optional): Url pointing to device provider site providing independent download bundle matching the MD5 checksum. This is necessary only for providers supporting their customers to download directly.

Version→docUrl: Url pointing to device provider site where RD service installation, upgrade, and usage documentation is provided.

Version→ProviderModel: This element may be repeated for all models that are supported by this RD Service version.

ProviderModel→mi: Provider Model ID. Alpha numeric.

ProviderModel→type: Type of the device. This is a comma delimited string depicting that this device can be used for fingerprint and/or iris and or photo (face modality not yet supported). Typically it will be just “F” or “I” or “P”. But if a device model supports both Fingerprint and Iris (unlikely scenario), this will be “FI” and so on.

ProviderModel→level: Certification level for this registered device model. It can be either “L0” or “L1”.

**/

5 Device Discovery

5.1 Linux & Windows Discovery & API Calling

The RD service will run on the host machine. The RD service will be listening on the port starting 11100 - 11120 binding only to 127.0.0.1. Any of these ports can be used by the RD service and as a best practice please start from the start to listen until you find one of these ports free. Applications should scan these ports and discover the service.

The RD service will respond back only for the queries listed as part of the interface. RD service should not entertain any other calls and should reject or disconnect without any indications. For all other failed/malformed request the RD service would simply not respond.

The RD-Service call is similar to a HTTP 1.1 GET call. Except that it use the RD-SERVICE as the verb and * as the request URI. A new VERB is introduced to avoid any HTTP based bots or virus from infecting the service (This is just based on the defense in depth). This solution all together cannot protect against all attacks but helps to ensure that the client is well aware of the situation.

Applications should first scan the ports and try call the following to see if there is a valid RD Service listening on the port.

RDSERVICE * HTTP/1.1

HOST: 127.0.0.1:<apps port>

EXT: app_name

The response is as follows:

```
HTTP/1.1 200 OK
CACHE-CONTROL:no-store
LOCATION:http://127.0.0.1:<rd_service_port>
Content-Length: length in bytes of the body
Content-Type: text/xml
Connection: Closed
<RDService status="READY|USED|NOTREADY|.." info="provider info for display purposes">
    <Interface id="CAPTURE" path="/rd/capture" />
    <Interface id="DEVICEINFO" path="/rd/info" />
    <Interface id="POSTAUTH" path="/rd/postauth" />
</RDService>
```

NOTE: POSTAUTH is OPTIONAL for RD service providers. Based on customer needs on the field, they may provide this.

Based on the path, subsequent calls from application can be made to RD Service.

http://127.0.0.1:<port>/<CAPTURE-path>
http://127.0.0.1:<port>/<INFO-path>
http://127.0.0.1:<port>/<POSTAUTH-path>

The RD-Service call can be called by any number of client simultaneously and the system can respond back the same information for everyone calling. The RD service will ensure it's able to connect to the device and other error checks before it responds back to the call. The status should be "READY" if the device is ready. For all other errors please look at the spec.

Apps have to decide if they need L0 or L1 devices or could change the authentication to multifactor in case of L0.

CAPTURE - The capture call is a blocking call and only one client can call at any point in time.

```
CAPTURE http://127.0.0.1:<rd_service_port>/<CAPTURE_path>
HOST: 127.0.0.1:<port>
<PidOptions /> <!—see XML details in earlier sections -->
```

Response to the capture call is:

HTTP/1.1 200 OK
CACHE-CONTROL:no-store
LOCATION:http://127.0.0.1:<rd_service_port>/<CAPTURE_path>
Content-Length: length in bytes of the body
Content-Type: text/xml
Connection: Closed
<PidData /> <!—see XML details in earlier sections -->

DEVICEINFO – This call is responsible to provide the device info.

DEVICEINFO http://127.0.0.1:<rd_service_port>/<INFO_path>
HOST: 127.0.0.1:<port>

HTTP/1.1 200 OK
CACHE-CONTROL:no-store
LOCATION:http://127.0.0.1:<rd_service_port>/<INFO_path>
Content-Length: length in bytes of the body
Content-Type: text/xml
Connection: Closed
<DeviceInfo /> <!—see XML details in earlier sections -->

POSTAUTH – This call is optional to provide any feedback post authentication back to RD service to allow RD service to do additional utility functions such as doing an re-init, or rotating keys, or printing receipt etc.

POSTAUTH http://127.0.0.1:<rd_service_port>/<POSTAUTH_path>
HOST: 127.0.0.1:<port>

HTTP/1.1 200 OK
CACHE-CONTROL:no-store
LOCATION:http://127.0.0.1:<rd_service_port>/<POSTAUTH_path>
Content-Length: length in bytes of the body
Content-Type: text/xml
Connection: Closed

```
<PostAuth txn="" err="">
        <!-- txn is the transaction attribute for AUA -->
        <!-- err is exactly the err attribute of the Aadhaar authentication response -->
  <CustOpts>
      <!-- This is open for the device providers to pass their options. All parameters passed as part of CustOpts
      element must be declared during certification and well documented. Note that data passed as part of
      CustOpts MUST NOT violate UIDAI policies and device providers must take care in defining this to ensure
      no sensitive data is passed. -->
  <!-- repeating element -->
    <Param name="" value="" />
  </CustOpts>
</PostAuth>
```

- All connection are closed after the response.
- The RD service will allow only one capture call at any given point in time.
- In case a app calls capture when the RD service is in between the capture then it should return appropriate error code as per spec.

5.2 Android Discovery & API Calling

- RD service should do the following actions:
  - Fingerprint devices should register "in.gov.uidai.rdservice.fp.INFO" and "in.gov.uidai.rdservice.fp.CAPTURE"
  - Iris devices should register "in.gov.uidai.rdservice.iris.INFO" and "in.gov.uidai.rdservice.iris.CAPTURE"
  - **INFO** should return the same DeviceInfo XML when called.
  - **CAPTURE** should return the same PidData XML when called.
- Applications integrating on Android should do the following:
  - Browse providers and let user choose an appropriate RD service. Application may provide "remember default" and other options based on their needs.
  - Call "INFO" intent and verify the provider package against locally cached UIDAIRDServices registry to make sure only certified ones are being used.
  - Call "CAPTURE" intent to capture the encrypted biometrics.

5.3 Custom IPC Calls

It is mandated that the RD service MUST run as separate process.In case socket based or Intent/URL based integration is not available, the RD service may communicate using any IPC method with the Authentication Application. Refer to FAQ document published for RD service for approval and certification for these exceptions.

6 Keystore Security

In case of L1 the Keys are safely stored by the PCH (Pre-Certified Hardware). Please refer the L1 Pre-certified Hardware Certification section for further clarity.

In case of L0 the following security has to be in place. **When OS is giving you a keystore facility satisfying the following requirements, device provider should use that**.

1. Keystore file should be limited with read and write rights only for the user as whom the RD service runs and no other user accounts should have access to the file/store.
2. The RD service should run as a user account who does not have login privileges.
3. Keystore password has to be complex and auto generated. The following list of approaches are possible:
    a. A combination of random data, user credentials and device identification data -derived key using identities (MAC, bluetooth, hard disk serial number, processor id and other device id's) that exist within the system. The logic how key is derived using these values has to be obfuscated to avoid any possible security threats.
    b. The Key derivation logic should be in a compiled native machine dependent and can not be an open api.
    c. The password should be changed for every Key rotation.
    d. White cryptography to derive the password.
    e. The password should be more than 16 characters in length and should contain minimum of 3 special characters, small letters, capital letters and numbers
    f. A server side logic could be built to help with opening the keystore.
4. The permission to access keystore should be restricted to the RD service. Any violation of the same should be detected by the RD service and it should inform the server about such failures. This failure would be tracked as an incident by the device provider.

5. All type of access and access attempts to the keystore should have audit logs.
6. The private key should not be extractable (wrapped or direct)
7. It is mandated that key pair is generated inside the capture_n_sign service. Note that device authentication must be performed before allowing any connection to management server,
8. The keystore has to be cleared and zeroed in case the RD service is deleted.

7 Register & DeRegister API

Device provider backend "Management Server" should call UIDAI register API whenever a new device needs to be registered. Device management front end to management server interfaces are specific to the device provider. Management requirement is specified in the next section.

This API will be whitelisted only for the device providers. Digital signature validation will be done for the callers to ensure only certified providers are able to call this API. In addition, IP whitelisting will also be done.

7.1 Register API

https://rdprod.uidai.gov.in/register

Input:

<RegisterDevice ver="" ts="" txn="">

<Device dpId="" dc="" mi="" idHash="" PCHCertificate**=”**<!-- public certificate of chip, issued by Pre certified Hardware Provider , **this need to be send only in case of L1**--> **”** />

<Signature/> <!-- digital signature of the provider -->

</RegisterDevice>

In case of L1 idHash has the below composition

IDHASH=concat <deviceid> + _##_ + <base64(Sign1)>

Where sign1= DigSign(TD) – Signed using PCH signing method
Where TD = deviceid:base64(<deviceid>);timestamp:<timestamp>

<deviceid> **Serial number of the device that can be used to physically identify the device.**

<PCHCertificate> - This element is not required in case of L0 Device Registration.

In case of L0 idHash must be SHA-256 of device serial number, that is used to recognize physical device.This should be read automatically without any user input. This ID is not expected to change during the lifetime of the physical device. idHash MUST match what was sent during registration

Output:

```
<RegisterDeviceResp ts="" txn="" code="" err="">
  <Signature/> <!-- digital signature of UIDAI -->
</RegisterDeviceResp>
```

7.2 DeRegister API

https://rdprod.uidai.gov.in/deregister

Input:

```
<DeRegisterDevice ver="" ts="" txn="">
  <Device dpId="" dc="" mi="" />
  <Signature/> <!-- digital signature of the provider -->
</DeRegisterDevice>
```

Output:

```
<DeRegisterDeviceResp ts="" txn="" code="" err="">
  <Signature/> <!-- digital signature of UIDAI -->
</DeRegisterDeviceResp>
```

8 Management Section

8.1 Management Client Specification

1. Management client may or may not be packaged with RD Service as an installable.
2. Management client should implement an "init" method internally to check if device is registered, connect to management server, initialize and rotate keys, and check for software upgrades.
3. When running, management client should detect for physical device connected and readiness of it.
4. If device is not registered, it should auto initiate registration.
    a. Management client should authenticate the device to ensure it belongs to the device provider using combinations of serial numbers, internal identifiers, signatures, etc. Internal ID that is used to recognize physical device (such as serial number) should be read automatically without any user input. This ID is not expected to change during the life of that physical device.
    b. In addition, for L0 devices, to avoid invalid/non-genuine devices being registered, a concept of "activation code" could be used to authenticate if the device is genuine.
        i. Device providers can send activation codes to people/entities who procure the device.
        ii. This provides a mechanism to do out of band authentication. In the case of L1, this is not required as signature from TEE is used along with registration.
        iii. Once it is activated, optionally user registration can be done and user authentication may be used for all management services in addition to client software authentication.
    c. Registration should include internal ID (serial number or any other internal ID that is used to recognize physical device), host fingerprint, timestamp, device keys, and other device details for authentication, etc.
    d. Device provider may keep additional attributes/info for their own management and audit purposes.
    e. Device provider should check pre-existence of serial number or other physical unique attributes to ensure same device gets same device code UUID. In the case of new registration, server generates a new device code (UUID) following **RFC4122 Version 4 standard for UUID** and should send back to client.

f. Device provider backend should call UIDAI Register service to ensure device is registered with UIDAI.
g. After successful registration with UIDAI, device provider backend should sign the device public key and return to client.

5. If device is registered, it should initiate key rotation when necessary.
   a. Management server should trigger key rotation under 2 scenarios:
      i. based on the trigger from RD service during "init" as per UIDAI key rotation policy;
      ii. based on the manual trigger from management client UI (this is needed only in special conditions where manual key reset needs to be triggered). This trigger should call same "init" to re-initialize.
   b. When key needs to be rotated, device should generate new key pair, send public key to server for signing and updating management server registry.
   c. Private key must be stored securely within keystore (L0) or within TEE (L1).
   d. See keystore security section above for details on keystore protection.
6. Management client should check for software upgrades and initiate upgrades.

8.2 Management Server Specification

1. All management server communication must be via HTTPS.
2. Management server should authenticate management clients and allow registration, key rotation, triggering upgrades, and other necessary management services. See previous section for details.
3. Management server should use HSM for Device Provider management.
4. Device database, secret token for authenticating management client, device fingerprint, user credentials, etc. should be protected through controlled access, encryption, or other security best practices.
5. Appropriate security mechanisms should be in place to protect HSM and device database access.
6. Log files should not contain any sensitive data.
7. Management server should implement configurable key rotation policies and should be configurable as per UIDAI policies.

9 L1 Device Addendum

9.1 L1 Certification Steps

The certification of L1 registered device consists of the following steps

1. "Pre-certified" hardware (PCH) validation by UIDAI/STQC
    a. Secure crypto block with international certifications
    b. TEE certification as defined in section "L1 Compliance" of this document.
    c. Secure provisioning process
2. L1 compliant registered device solution architecture validation by UIDAI/STQC using "pre-certified" hardware
    a. Secure system design inline with the key objectives of the UIDAI RD Service specification (latest version)
    b. Implementation of RD Service and Management Client inline with RD Service Specification (latest version)
3. STQC certification process
    a. Functional test - Similar to L0 certification
    b. Compliance test - Similar to L0 certification
    c. FRR test - Similar to L0 certification
    d. Security test. - Combines L0 and L1
    e. Demonstration of system level tamper responsiveness (if applying for tamper responsiveness)

9.2 L1 Device Threats:

The following set of threats are expected to be addressed as a part of the L1 device solution architecture. Its expected that the pre-certified hardware provider would provide documentation to support the solutions claims on mitigating these attacks. The mitigation techniques if listed in this document are standard choices and vendors may implement other techniques to mitigate threats.

9.2.1 "Pre-certified" hardware (PCH), system software threats

The Pre-Certified hardware, system software should protect against the following threats.

1. Hardware cloning attack

2. Hardware Tamper attacks
    a. Physical tamper, voltage, frequency, temperature attacks on crypto block
3. Differential Power analysis attack
4. Probing attacks
5. Segregation of memory for execution of cryptographic operation (crypto block should be protected from buffer overflow type attacks)
6. Vulnerability of the cryptographic algorithm implementation
7. Attacks against secure boot & secure upgrade
8. TEE/Secure processor OS attacks

9.2.2 L1 Registered Device System Level threats

1. Probing attacks: Connectivity from the sensor to pre-certified hardware. One or more of the following mitigations may be implemented to achieve the tamper responsive at the system level, i.e keys must be erased if a tamper is detected.
    a. Protective mesh
    b. Minimize exposed surface between sensor and the pre-certified hardware
    c. Encrypted channel between sensor and pre-certified hardware
2. Software protocol attacks: All of the following mitigations must be implemented for a L1 certification.
    a. Limit the maximum size for the communication from host to device to prevent communication channel from buffer overflow/stack overflow attack.
    b. Forcible software updates for security vulnerability
    c. Minimize the protocol usage to the following
        i. Capture, Device Info, Key rotation, Time Sync, Upgrade, Register, Update UIDAI certificate
    d. Declare any other functions implemented

9.3 "Pre-certified" hardware, system software certifications/validations:

The following certifications will be used for hardware, system software "pre-certification"

1. Secure crypto block hardware certifications may use ONE of the following certifications
    a. FIPS 140-2 Level 2 (tamper evidence) FIPS 140-2 Level 3 (Tamper resistance) - This covers hardware and software and all form of attacks.

b. PCI - PTS v4.1 and above Pre-Certified - This covers both physical tampering and software
c. PCI - PED 2.0 Pre-Certification and above
d. One of following Common Criteria (CC) certification or equivalent custom profiles at level EAL4 and above
    i. https://www.commoncriteriaportal.org/files/ppfiles/pp0035a.pdf
    ii. https://www.commoncriteriaportal.org/files/ppfiles/pp0084a_pdf.pdf

2. Cryptography Algorithm Certifications:
    a. CAVP validation (If not covered in the above certifications)
        i. RSA, AES, SHA-256 (Running on secure cryptoprocessor)
        ii. TRNG Certification (DRBGVS or equivalent)
    b. FIPS 140-2 Level 1 (only for cryptography algorithm if not covered in the above certifications)
        i. RSA, AES and SHA-256
        ii. TRNG Certification
3. Self Certification:(if the design does not have a certified secure crypto block)
    a. Detailed documentation and lab reports about the implementation of "protection of side channel attacks for cryptographic operations".
    b. Self certification that the solution ensure to zero (or any other technique to ensure that the data can not be recovered) all used memory (except the CIk) upon detecting a tamper attempt during the ON state.
    c. Self certification that the PCH does not store unencrypted keys in non volatile memory at any time.
4. TEE certification.
    a. Global platform certified TEE is required in the case of shared hardware i.e the processor used for purposes other L1 Registered device functionality
    b. In the case of dedicated hardware for L1, proving secure boot, secure upgrade and isolation of access to the secure crypto block is required for the purpose of L1 certification. The following test cases should be demonstrated (if not included in the third-party international certification) for "pre-certification" on a development board including all hardware (except the biometric sensor):
        i. Secure boot
            1. Should check for the integrity of the hardware platform upon every boot. The hardware configuration should be identical to the configuration at first boot.
            2. Should validate the signature of the boot image upon every boot.
        ii. Secure upgrade of OS/crypto block

1. Has to ensure that there is no boot possible when an upgrade process fails. The provider should rollback to previous known version or re-attempt upgrade (max 10 attempts)
2. Forced upgrade in case of vulnerability detection

iii. Secure Upgrade of Device Provider Application

9.4 Identity for Pre-Certified Hardware:

Identifying a pre-certified hardware is one of the key part of the L1 compliance. The following section describes the compliance needed to create the identity into the secure crypto-block. The device identity is strongly tied to the cryptoprocessor to ensure non-clonability.

1. The identity is based on a RSA 2048 bit key pair generated within the secure crypto block and the public key is signed by pre-certified hardware provider (at the pre-certified hardware provider or at programming house working with the pre-certified hardware provider).
2. The identity is created by generating a key pair in the secure crypto block and public key is signed by the PCH. This signed public certificate should be permanent and written to OTP (One Time Programmable) memory. The chip should live and die with this single identity.
3. The identity of the pre-certified hardware is as described below.
    a. A random RSA 2048 bit key pair is created and a CSR for the same is generated.
    b. The CSR is used to issue a chip identity (x509 certificate). The parent certificate would be same as the pre-certified provider's root of trust. This new issued certificate is called CIk which is stored in One Time Programmable (OTP) memory
    c. The pre-certified hardware would have the ability to sign the device identity "TD" using the identity key.
4. No other identity keys should be present in the pre-certified hardware at this stage other than the pre-certified hardware identity key
5. Root of trust for the pre-certified hardware and device provider root should be provisioned on Read Only Memory (ROM) or One Time Programmable (OTP) memory
6. All the PCH signing key pairs used during the identity creation process should be (generated and stored) in a FIPS 140-2 Level 3 device and its the responsibility of the PCH vendor to ensure safety of these keys.

9.5 Secure Boot and Secure Upgrade

The device should have the ability to boot and upgrade securely. (Except for a thin secure boot layer stored on OTP memory, all other firmware and software should capable of being securely upgraded.) The following points would describe the minimum compliance for the same.

1. All software loaded to the chip should be verifiable with cryptographically safe hash (SHA256) and signatures.
2. The root of trust for all signatures (both device provider and pre-certified hardware vendor) would be uploaded during the identity creation for PCH process.
3. The boot sequence should check for the integrity of the softwares using the respective provider keys. It is expected that
    a. Upon 10 (10 is the max limit and vendors can choose a lower threshold) failed attempts to boot (first boot) after upgrade process of the device, all the device software, data, license keys, keys used for biometric signing and any other keys used by the device provider except keys related to chip identity and root of trust should be deleted. The device is now considered to be in the tampered state. Field upgrade of a tampered device is not allowed, it must be re-programmed in the factory. (device provider must demonstrate secure method for reprogramming)
    b. Upgrade of firmware will always validate the root of trust & should move upward in version number. So firmware downgrade should be impossible.
4. . The system cannot be functional in a partially upgraded state..
5. In the event of failure in upgrade, the secure boot should re-validate the signatures.

9.6 Secure Provisioning

1. The secure provisioning facility should be auditable upon need.
2. The key creation should happen over a secured facility and no users should have access to, influence or steal the identity.
3. The provisioning should happen over secure & encrypted channel with minimum of 2048 bit of RSA or equivalent (more) PKI & a FIPS 140-2 Level 3 device.
    a. CI_k Should be created during this process
    b. The root of trust certificate of the pre-certified hardware , Chip Identity Certificate (CI_k)and root of trust certificate of device provider should be written in Read only / OTP memory
    c. The secure boot manager is loaded

4. All debug options should be blocked on the pre-certified hardware as part of the provisioning process.

9.7 Hardware/System Software Vendor Self Certification:

1. No backdoors or debug mode enabled on the secure crypto-block
2. Documents/certifications submitted are valid for the current model of secure crypto-block.
3. The secure provisioning process is in compliance with global security standards and UIDAI specification
4. The solution architecture submitted by the device provider is inline with the security guidelines recommended by the pre-certified hardware provider
5. System level tamper responsiveness implementation by the device provider has been validated (if applicable)

9.8 System Level Tamper Responsiveness Certification

It is required to minimize the attack surface at the system level by using methods such as but not limited to hidden traces, protective meshing, encrypted communication etc. Minimizing the attack surface is inline with the objective 1 of this document.

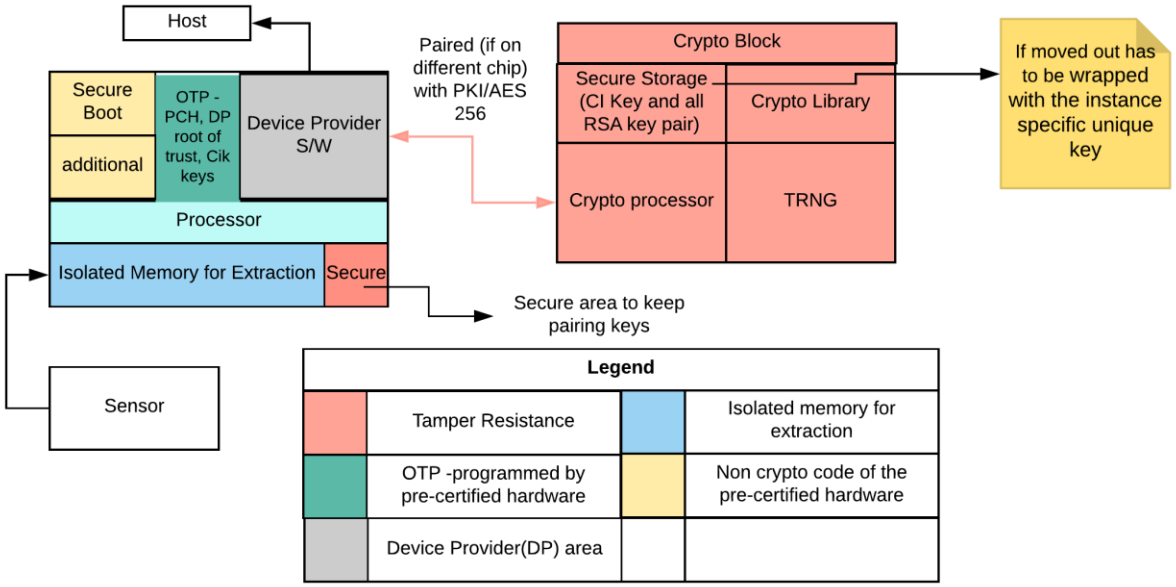
If certification for tamper responsiveness is requested the vendor must submit the various tamper responsive measures implemented.

1. Review the design to understand approach to system level tamper including some or all of the following mitigations
    a. Protective Meshing
    b. Box open Tamper
    c. Pairing based Tamper Responsiveness
    d. Chemical tamper responsiveness
2. Test Cases to demonstrate the Tamper Responsiveness. PCI PED compliance test can be used as a guideline. PCI PED compliance certification is not expected.

    UIDAI/STQC will differentiate between devices with and without tamper responsiveness in the certification process.

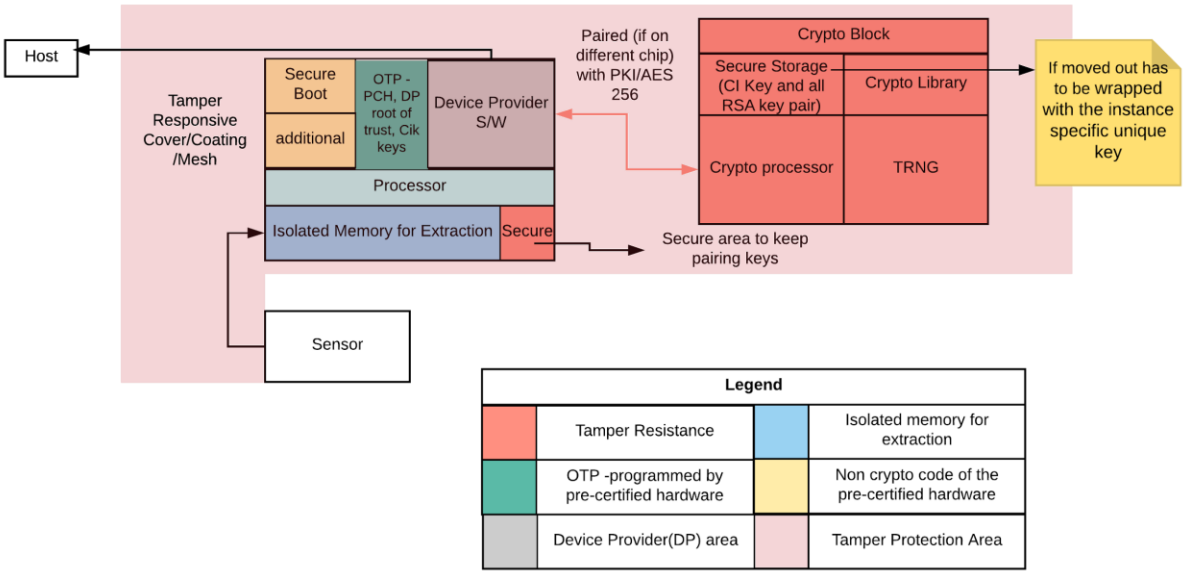
9.9 Reference Design for L1

L1 without system level tamper



Reference Design for L1 without System Level Tamper

L1 with system Level tamper



Reference design for L1 with System Level Tamper

Note: The diagrams are for logical illustrations and the actual implementation would be different. The images are in 2D and the protection has to be applied on a 3D object.

10 Error Codes

10.1 Error codres for RD Service

100 “Invalid PidOptions input. XML should strictly adhere to spec.”
110 “Invalid value for fType”
120 “Invalid value for fCount”
130 “Invalid value for iType”
140 “Invalid value for iCount”
150 “Invalid value for pidVer”
160 “Invalid value for timeout”
170 “Invalid value for posh”

180 “Face matching is not supported”

190 “Invalid value for format”
200 “Invalid Demo structure”
210 "Protobuf format not supported"
700 “Capture timed out”
710 “Being used by another application”
720 “Device not ready”
730 “Capture Failed”
740 “Device needs to be re-initialized”
750 "RD Service does not support fingerprints"
760 "RD Service does not support Iris"
770 "Invalid URL"
999 “Internal error”

10.2 Error Codes for Register API

100 “Invalid XML format”
110 “Invalid XML Version”
120 “Invalid timestamp”
130 “Timestamp should not be older than <10 minutes>”
140 “Invalid DPID”
150 “Invalid MI”
160 “Digital Signature Validation Failed”
170 “Device Already Registered”
180 “ PCH Certification Validation Failed”
190 “IdHash Validation Failed”
200 “Duplicate Device Serial Number in IdHash”
999 “Unknown Error”

10.3 Error Codes for De-Register API

100 “Invalid XML format”
110 “Invalid XML Version”
120 “Invalid timestamp”
130 “Timestamp should not be older than <10 minutes>”
140 “Invalid DPID”
150 “Invalid MI”

160 “Digital Signature Validation Failed”
170 “dc already de-registered”
180 “dc not available to de-register”
999 “Unknown Error”