Unique Identification Authority of India (UIDAI)

3rd Floor, Tower II, Jeevan Bharati Building,

Connaught Circus, New Delhi 110001

AADHAAR REGISTERED DEVICES

TECHNICAL SPECIFICATION - VERSION 2.0 (REVISION 1)

February 2017

Contents

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 http://authportal.uidai.gov.in

1.2 Target Audience and Pre-Requisites

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 http://uidai.gov.in/images/FrontPageUpdates/aadhaar_authentication_api_2_0_1.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 UDIAI 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.

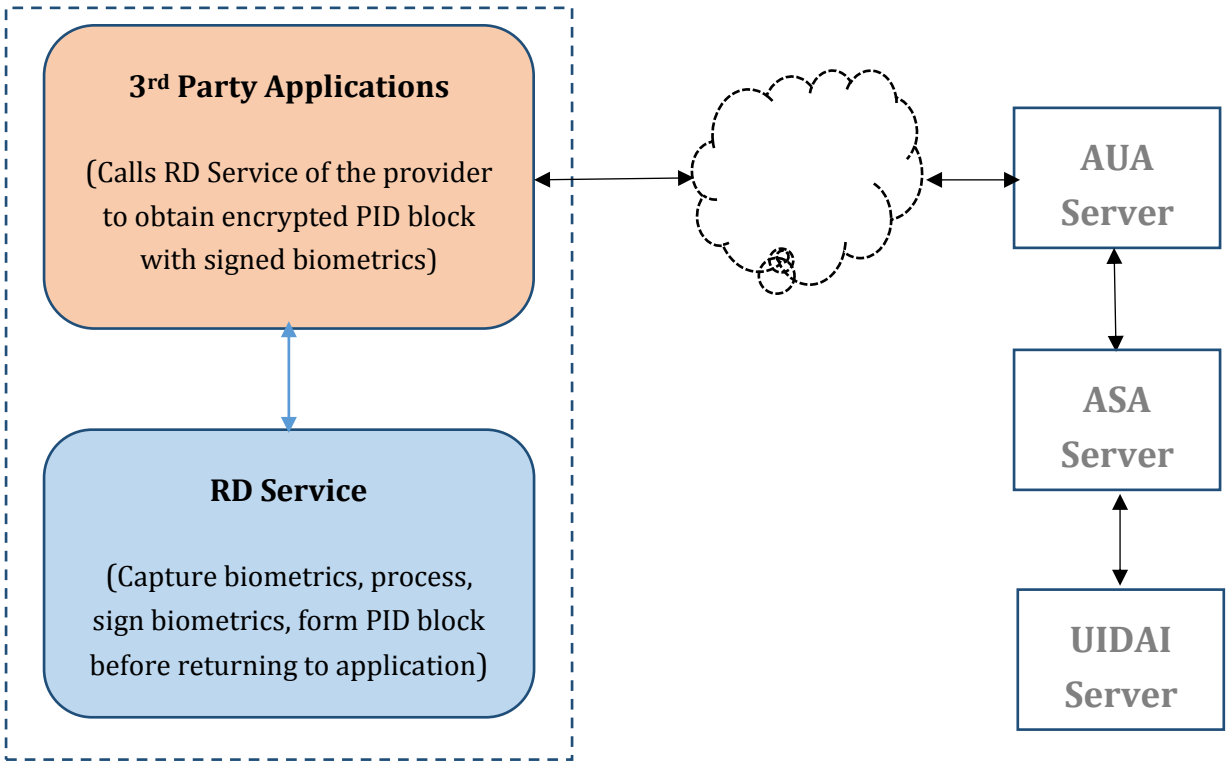
2.2 Registered Devices

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 provider 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 provider 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 2.0 specification for details).

2.3 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 either generate an asymmetric key pair within the device (highly recommended) or securely initialize the key pair. This would be the device key pair. Every physical instance of the device should have its own device key pair.
5. Device public key should be signed by one of the device provider key. Refer section 2.7 for supported algorithms. Provider can sign the public key either within the device or on provider server over a secure channel during key initialization.
6. Device provider MUST ensure each physical device has a unique code. Maximum length of the code is 50 characters when represented as string. To ensure device codes are globally unique it is necessary that device provider uses a 128-bit UUID (represented in HEX notation).

Note: Device public-private key generation/initialization 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 *Meta* element of authentication XML (see Authentication API Specifications 2.0).
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 metadata 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: `SHA-256(dpId+rdsId+rdsVer+dc+mi+idHash)`
    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 – 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).
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 2.0 specification and returns it to application along with other metadata.
    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.

2.4 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**.

2.4.1 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.

2.4.2 Input/Output XMLs

```
<PidOptions ver="">
    <Opts fCount="" fType="" iCount="" iType="" pCount="" pType="" format="" 
          pidVer="" timeout="" otp="" wadh="" posh=""/>
    <Demo></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: Version of the PidOtopns spec. Currently it is "1.0". This is necessary to 
           allow applications to gracefully upgrade even when RD service may be been 
           upgraded. RD Service must support current version and one previous version 
           to allow apps to upgrade at different points in time.

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 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.

*/



<PidData>

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

    <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 </pid>

</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="" />

/*

      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.

*/
```

2.5 Levels of Device Compliance

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

2.5.1 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.

2.5.2 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.

2.6 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.

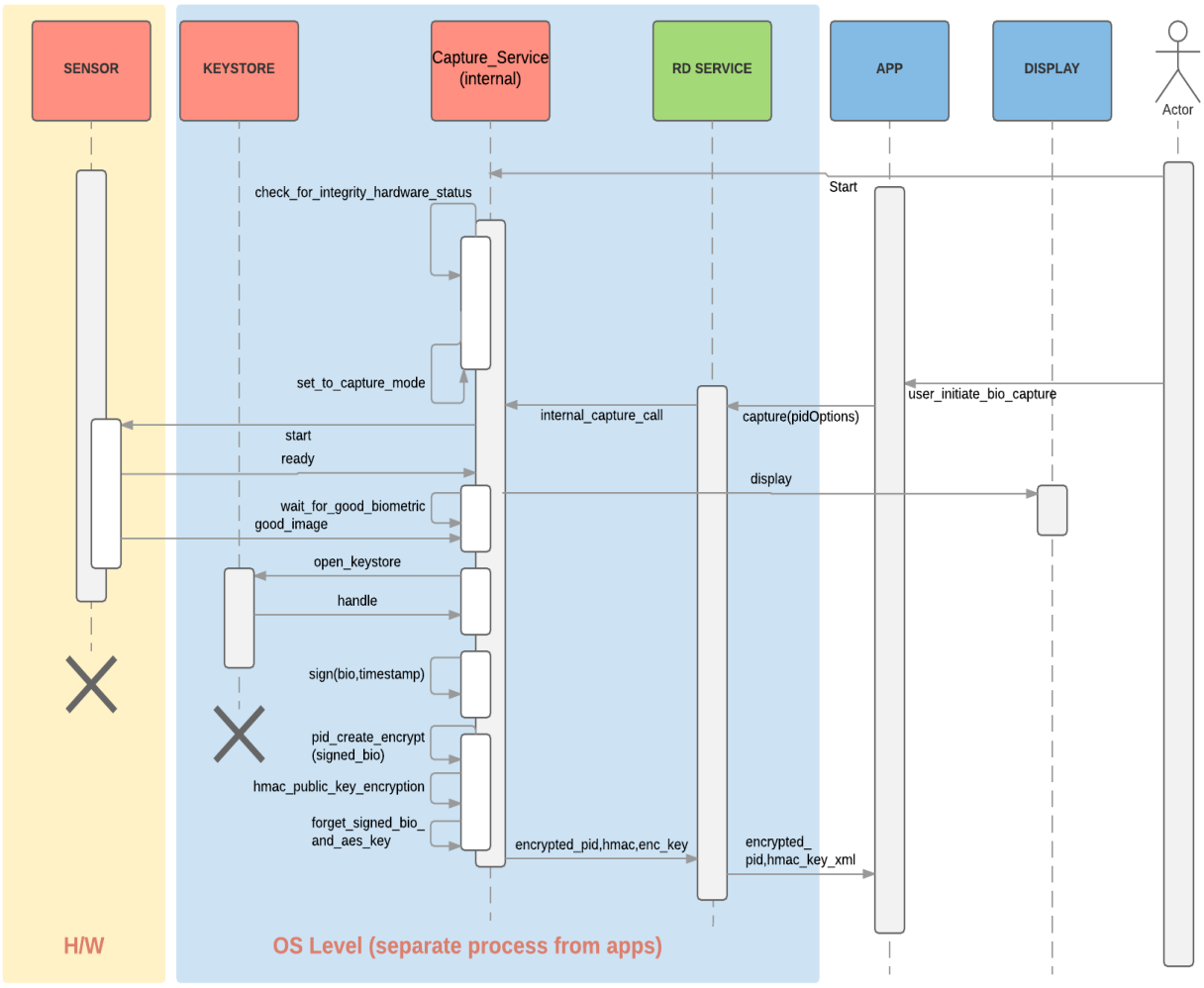
2.7 Certificates, Keys Policies

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.

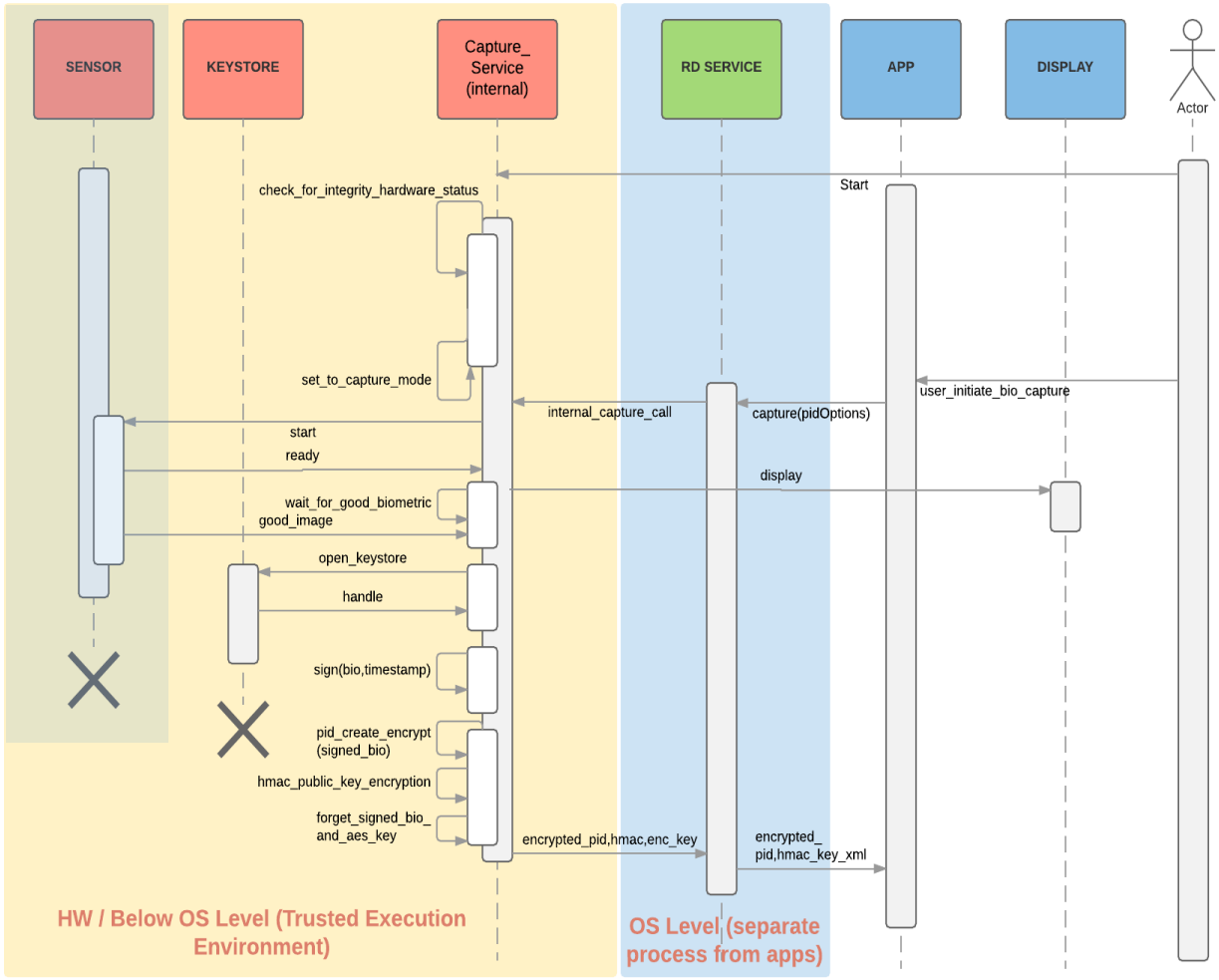
5. Device provider SHOULD have necessary server/backend infrastructure to sign the device public key, rotate device keys older than specified time as per UIDAI policy, update UIDAI public key used for encryption, and provide updates/fixed to their RD Service.

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.

Certified RD services and devices details are made available at the following URL: `https://authportal.uidai.gov.in/devices/rdservice_registry.xml`

Beta version details that are going through certification is made available at: `https://authportal.uidai.gov.in/devices/rdservice_registry_beta.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="">
             <!-- 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 can be made 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→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. Thi 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 with on the host machine. The RD service will be listening on the port starting 11100 - 11120 biding 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. 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-cache
```

```
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" />
</RDService>
```

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>

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-cache
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.

```
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-cache
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 -->
```

- 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 name against locally cached `UIDAIRDServices` registry to make sure only certified ones are being used.
- Call "`CAPTURE`" intent to capture the encrypted biometrics.

6 Keystore Security

In case of L1 the following security has to be in place.

1. Keystore is inside the TEE and only the internal service has access to the same.
2. The private key should not be extractable (wrapped or direct)
3. The key pair has to be generated inside TEE.

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, harddisk 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 to help with opening the keystore.
4. The RD service should fail its integrity check upon the keystore file permissions not correct or has unwanted access and 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 recommended that key pair is generated inside the capture_n_sign service. If key pairs are generated on management server, then private key must be returned and MUST BE strongly encrypted using AES-256 session key generated at the client. 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. Both digital signature and API key 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://authportal.uidai.gov.in/rd/service/register
```

Input:

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

Output:

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

7.2 DeRegister API

```
https://authportal.uidai.gov.in/rd/service/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 authentication 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 if 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) 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 service should trigger key rotation under 2 scenarios:
        i. based on the trigger from management server during "init" (ideally done at least once a day);
        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.