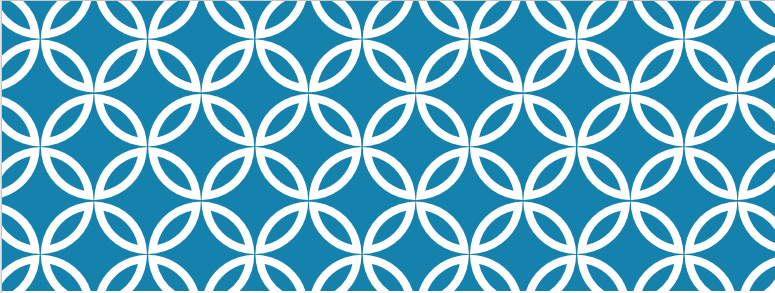
a



MOSIP

Modular Open Source Identity Platform

Version 0.1 | 24 Aug 2018

Packet Creation - LLD

Copyright Information

This document is the exclusive property of \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_; the recipient agrees that they may not copy, transmit, use or disclose the confidential and proprietary information in this document by any means without the expressed and written consent of Mindtree. By accepting a copy, the recipient agrees to adhere to these conditions to the confidentiality of \_\_\_\_\_\_\_\_\_\_\_\_\_ practices and procedures; and to use these documents solely for responding to \_\_\_\_\_\_\_\_\_\_\_\_\_\_ operations methodology.

Revision History

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Ver** | **Change Description** | **Sections** | **Date** | **Author** | **Reviewer** |
| 0.1 | First Draft | All | 24-Aug-18 | Omsaieswar Mulakaluri | Karthik R |
| 0.2 | Second Draft | All | 29-Aug-2018 | Omsaieswar | Karthik R |
| 0.3 | Third Draft | Packet Structure | 06-Sep-2018 | Omsaieswar | Karthik R |

References

|  |  |  |  |
| --- | --- | --- | --- |
| No | Document Name | Ver. | Location |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Glossary

|  |  |  |
| --- | --- | --- |
| **Terminology** | **Definition** | **Remarks** |
| EC | Enrollment Client / ID Issuance Client application. |  |
| IDC | ID Issuance Client | Enrolment client |
| EO | Enrollment Officer |  |
| ES | Enrollment Supervisor |  |
|  |  |  |

Table of Contents

[Copyright Information 2](#_Toc524000486)

[Revision History 2](#_Toc524000487)

[References 2](#_Toc524000488)

[Glossary 2](#_Toc524000489)

[Table of Contents 3](#_Toc524000490)

[Part A: Background 5](#_Toc524000491)

[1 Introduction 5](#_Toc524000492)

[1.1 Context 5](#_Toc524000493)

[1.2 Purpose of this document 5](#_Toc524000494)

[2 Scope 5](#_Toc524000495)

[2.1 Functional Scope 5](#_Toc524000496)

[2.2 Non Functional Scope 6](#_Toc524000497)

[2.3 Assumption 6](#_Toc524000498)

[2.4 Out of Scope 7](#_Toc524000499)

[3 Technical Approach 7](#_Toc524000500)

[3.1 Design Detail 7](#_Toc524000501)

[3.1.1 Packet Structure 9](#_Toc524000502)

[3.1.2 Folder level Data: 10](#_Toc524000503)

[3.1.3 Entity Object Structure: 11](#_Toc524000504)

[3.1.4 Validations: 11](#_Toc524000505)

[3.2 Class Diagram 11](#_Toc524000506)

[3.3 Sequence Diagram 11](#_Toc524000507)

[4 Success / Error Code 12](#_Toc524000508)

[5 Dependency Modules 12](#_Toc524000509)

[6 Database - Tables 13](#_Toc524000510)

[7 User Story References 13](#_Toc524000511)

[8 Pending Items / FAQ 13](#_Toc524000512)

Part A: Background

# Introduction

## Context

MOSIP is developed as an open source framework project. The java standard design principles will be followed to design the component.

## Purpose of this document

This document provides the low level technical design approach of a particular functionality in MOSIP Platform. It details out the in depth technical area of a particular scope.

# Scope

## Functional Scope

* Expose the API to create the Enrollment packet, where the data captured in the ID Issuance client UI application should be stored.
* Packet should have the detail of:
  + Applicant – Demo and Bio-metric
  + Introducer – Demo and Bio-metric
  + HOF – Demo and Bio-metric
  + Officer / Supervisor – Bio-Metric
  + Enrollment Id.
  + Packet Metadata.
  + Enrollment Acknowledgement form.
* Enrollment packet should be stored in encrypted format in the local hard disk.
* Once the packet is created the same shouldn’t be sent to the server until it is approved by the Enrollment Supervisor.
* The API should return the success / failure status code along with the respective message.

## Non Functional Scope

* Security :
  + The Enrollment packet shouldn’t be decryptable other than Enrollment Server.
  + Hash out the data – the hash code of the data should be sent along with the packet.
  + Along with the packet, the hash should also be using RSA public encrypted.
  + Un-encrypted data shouldn’t be stored in local hard disk during the creation of Enrollment packet.
  + The IDIS application able to get the RSA public key from Core Kernel module.
  + The IDIS able to generate the AES key seed [256-bit] by using the EO ID, MAC of the machine and timestamp.
* Log the each state of the packet creation:
  + As a security measures the UIN or customer information should not be logged.
* Cache :
  + Enrollment packet data shouldn’t be cached and clear off all the data from the JVM local memory once the packet is created in local hard disk.
* Audit :
  + Each state of the packet creation should be stored into the DB for audit purpose.
  + UIN and important detail of the customer should not be audited.
* Exception :
  + Any exception occurred during the packet creation the same will be reported to the user with the user understandable exception.
* Data History :
  + The IDIS able to authenticate by using the Core Kernal module.
  + Maintain the Enrollment id, status and other high level info in the database table.
* Configuration:
  + Public Key – the respective byte values will be present in the database table along with the expiry detail.
  + Before initiating the enrollment process, the key expiry to be validated.

## Assumption

* System should have enough space to create the packet at the desired location.
* The valid Key file [public] will be present in the client machine.
* This API should get the captured fingerprint, iris and authenticated [officer biometric] information as an images bytes from the UI client.
* The UI client should follow the defined [DTO structure](#_Entity_Object_Structure:) while capturing the Enrollment data from the Resident.
* The Enrollment ID will be generated before invoking this Packet creation process.
* Every officer will have their own ID that will be used during AES seed creation.

## Out of Scope

* Validation of system space while creating the packet.
* RSA Key pair generation and management.
* Send the packet to the Enrollment server.
* Archival of the Enrollment packet.
* Enrollment ID creation.
* Validating the request object provided by the UI client. [will be taken care in the next sprint]

# Technical Approach

## Design Detail

The detailed technical process for Enrollment packet creation is provided below:

**Packet API:**

* Create a Java component as ‘PacketHandler’ with ‘createPacket’ method to accept the Defined [DTO structure](#_Entity_Object_Structure:) from the client application.
* Validate the request object against the Business rule and render the respective error message [user defined] to the invoking client application if any rule failed. [this activity will be taken care in the next sprint]
* If the provided request object is valid then continue with the rest of the process.
* Prepare the Zip object, which is to be stored into a configured location.
* Get the Demographic byte stream from the respective DTO object and store it into the Zip object using right folder path [….].
* Get the Biometric byte stream from the respective DTO object and store it into the Zip object using right folder path [….].
* Get the Proof of documents byte stream from the respective DTO object and store it into the Zip object using right folder path [….].
* Get the ‘Enrollment ID’ from the respective request object, write the same into the File object and save the file object into the Zip object.
* Hash :
  + Generate the Hash for the Biometric, Demographic and EID of Resident Information.
  + Use the HMAC generation from Java 8 [MD5 Hashing – SHA256]
* Store the generated Hash in a file and append to the created Zip object.
* Capture the Enrollment Officer/Supervisor Authentication finger image from the respective DTO object and append to the Zip object.
* Create the Packet Info JSON file, which contains the **Meta data** information about packet and appended to the existing Zip object.
* Session Key Encryption:
  + Session key generation is [MAC of machine + EO Id + Timestamp] should not exceed 32 characters.
  + Pass the created Zip object [in-memory] through the AES-256 bit encryption.
  + Pass the Random Session Key as a seed to this AES encryption.
  + Get the Enrollment Officer Id from user context object.
* RSA Public Key Encryption:
  + AES Session key bytes pass through the RSA public key encryption.
* Use the “#KEY\_SPLITTER#” as a key separator for the AES encrypted bytes and the RSA Public key encrypted Session key seed.
* Append the RSA Public key Encrypted Session Key, Key Separator to the AES encrypted bytes.
* Save the encrypted data as a ZIP in local file system under the defined location in configuration file.
* Append the EO and machine information as a META-INFO JSON file and create another ZIP out of it. [Packet Zip + META-INFO JSON]
* Audit the exception/start/exit of the each stages of the packet encryption mechanism using AuditManager component.
* The final zip name should be as enrollemntid+CurrentTimestamp [28 digit].
* Timestamp format is [DDMMYYYYHHMMSSS]
* Once the packet has been successfully created then update the packet information in the ‘Enrollment’ table.

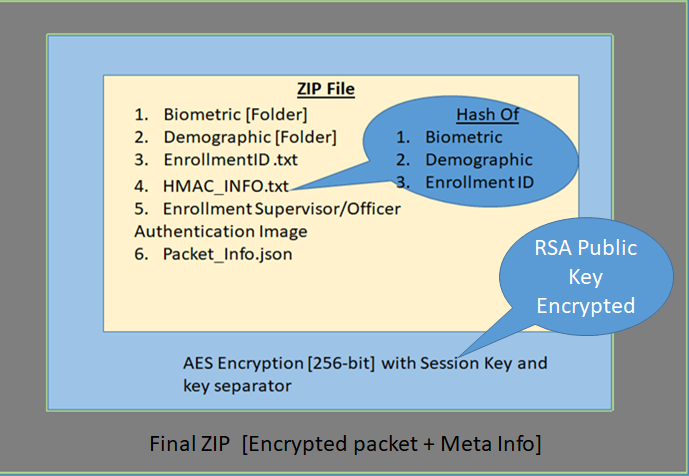
**Client [UI] Application:**

* Invoking client application should store all information about the resident as desired format of [DTO objects](#_Entity_Object_Structure:).
* Enrollment ID should have already been generated and pass it in the EnrollmentDTO object.
* Invoke the ‘PacketHandler’.createPacket(EnrollmentDTO) method to prepare the Enrollment packet at the configured location in local machine.

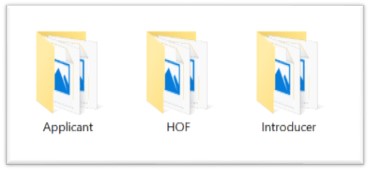
**Packet Archival:**

* Get the Packet status using the ‘Enrollment packet status’ reader REST service. If the status is UIN generated /Updated, we need to update the same info to the database and clean the packet. {will be taken care in the next sprint}

### Packet Structure

****

* Create date wise folder, if not exists. [Sample: 12-SEP-2018 ]
* Biometric and Demographic folders should have the below sub folder structure.
  + Applicant
  + Introducer
  + HOF
* **Biometric File:**



* **Demographic :**



### Folder level Data:

1. **Biometric**
2. Applicant **🡪** 
   * + - * LetThumb.jpg/png
         * RightThumb.jpg/png
         * LeftPalm.jpg/png
         * RightPalm.jpg/png
         * LeftEye.jpg/png
         * RightEye.jpg/png
3. HOF **🡪** 
   * + - * **HOF LeftThumb.jpg/png**
4. Introducer **🡪** 
   * + - * **LeftThumb.jpg/png**
5. **Demographic** 
   1. Applicant **🡪** 
      * + - ProofOfIdentity.docx
          - ProofOfResidenty.docx
          - ProofOfAddress.docx
          - ApplicantPhoto.jpg/png
          - ExceptionPhoto.jpg/png [If Exceptional cases]
          - Enrollment Acknowledgement.jpg
   2. Demographic\_info.json



1. **EnrollmentID.txt**
2. **HMAC File.txt**
3. **Packet\_MetaInfo.json**

****

1. **Enrollment Officer Bio Image[JPEG]**
2. **Enrollment Supervisor Bio Image[JPEG]**
3. **Meta\_Info.json [Outside of the encrypted Packet]**

****

### Entity Object Structure:

**Packet DTO Structure**

****

### Validations:

* Verify the Packet decryption, but this not in our scope but as a demo we need to show.
* The seed length should be 256-bit.
* The packet structure should be validated.
* The packet name should be unique and the name of the packet is[EnrollmentID+TimeStamp[DDMMYYYYHHMMSSS]]

## Class Diagram

[**https://github.com/mosip/mosip/blob/DEV/design/registration/\_images/\_class\_diagram/registration-packetcreation-classDiagram.png**](https://github.com/mosip/mosip/blob/DEV/design/registration/_images/_class_diagram/registration-packetcreation-classDiagram.png)

## Sequence Diagram

**<https://github.com/mosip/mosip/blob/DEV/design/registration/_images/_sequence_diagram/registration-packetcreation-sequenceDiagram.png>**

# Success / Error Code

While processing the packet if there is any error or successfully created the packet then send the respective Success or error code to the UI from API layer as Response object.

|  |  |  |
| --- | --- | --- |
| **Code** | **Type** | **Message** |
| 0000 | Success | Packet Successfully created |
| IDC-FRA-PAC-001 | Error | Unable zip the packet. |
| IDC-FRA-PAC-002 | Error | No socket is available |
| IDC-FRA-PAC-003 | Error | The host is unknown |
| IDC-FRA-PAC-004 | Error | No such algorithm available for input |
| IDC-FRA-PAC-005 | Error | No such padding available for input |
| IDC-FRA-PAC-006 | Error | Invalid key for input |
| IDC-FRA-PAC-007 | Error | Invalid parameter for the algorithm |
| IDC-FRA-PAC-008 | Error | The block size is illegal for the input |
| IDC-FRA-PAC-009 | Error | Bad padding for the input |
| IDC-FRA-PAC-010 | Error | Invalid seeds for key generation |
| IDI-FRA-PAC-011 | Error | IO exception |
| IDC-FRA-PAC-012 | Error | Exception while parsing object to JSON |
| IDC-FRA-PAC-013 | Error | Illegal key size for key generation |
| IDC-FRA-PAC-014 | Error | Invalid key spec for input |
| IDC-FRA-PAC-015 | Error | File not found for input path |
| IDC-FRA-PAC-016 | Error | Input-output relation failed |
| IDC-FRA-PAC-017 | Error | Class not found for input |

**Audit LOG:** Following status should be logged into the Audit Manager while processing the packets**.**

|  |  |
| --- | --- |
| **Type** | **Description** |
| Success | Packet Successfully created. |
| Encrypted | Packet Encrypted Successfully |
| Uploaded | Packet Uploaded Successfully |
| Synched to Server | Packet Synched to Server Successfully |
| Deleted | Packet Deleted Successfully |
| Approved | Packet approved Successfully |
| Rejected | Packet Rejected Successfully |
| Hold | Packet Hold on particular stage |
| Internal Error | Packet creation Error |
|  |  |

**DB Packet Table [Status code and description]:**

|  |  |
| --- | --- |
| **Code [Status\_Code]** | **Description** |
| C | Packet Encrypted and successfully created. |
| U | Packet Uploaded Successfully |
| S | Packet Meta information synched to server |
| D | Packet Deleted |
| A | Packet approved |
| R | Packet Rejected |
| H | Packet Hold on particular stage |
| E | Packet errors[ Ex : Virus scanner error] |
|  |  |

# Dependency Modules

|  |  |  |
| --- | --- | --- |
| **Component Name** | **Module Name** | **Description** |
| Audit Manager | Kernel | To audit the process while creating the packet. |
| Exception Manager | Kernel | To prepare the user defined exception and render to the user. |
| Log | Kernel | To log the process. |
| JOSN Utility | Kernel | To convert the object to JSON structure |
| ZIP Utility | Kernel | To convert the packet structure to ZIP |
| Encryption | Kernel | Encrypt the packet information using AES and RSA |
| Key Generator | Kernel | To get the generated public key |

# Database - Tables

1. PACKET – table

Structure:



# User Story References

|  |  |
| --- | --- |
| **User Story No.** | **Reference Link** |
| **MOS-64** | <https://mosipid.atlassian.net/browse/MOS-64> |
| **MOS-65** | <https://mosipid.atlassian.net/browse/MOS-65> |

# Pending Items / FAQ

1. Is there any new file will be appended to the encrypted packet, which contains the EO’s information?

Comment: Yes [Meta\_Info.json]

1. What is the maximum number of packets we can save in in-memory?

Comment: 1 [Because we may save the packet in to the machine location, and the offline save packet counts should be 1001 in offline]

1. If the System crashes, how we are going to recover these packets?

Comment: Consultancy has to inform the **Resident** to come over and submit once again

1. Is the supervisor immediately verify the resident information?

Comment: EOD process

1. Is the supervisor comes to each machine and verify the packets?

Comment: Yes

1. Enrolment ID generation and packet file name?

Comment: The packet file name is unique [Enrollment ID + Timestamp [DDMMYYYYHHMMSS]

Ex File Name: Enrollement ID\_TimeStamp

Length of Enrollment ID: 14 digit

Length of Timestamp: 14 digit.

1. UIN information stored in client DB?

Comment: No

1. Is there possibility that the Resident can enroll in one machine and go for EID correction or UIN updating in another machine?

Comment: Yes, the Resident can go but we need to maintain the enrollment ID/ UIN information as part of the “PacketMeta” info and need to maintain the packet status [ex: created/updated…]

File Name: New packet will be created with the updated information.

1. How and when to push the packet to the server to be decided? If we use the Export option then manual intervention in required to push the packet. So, raised the concern to get the update on this process..
2. Where to maintain the Public key provided by the Server application?