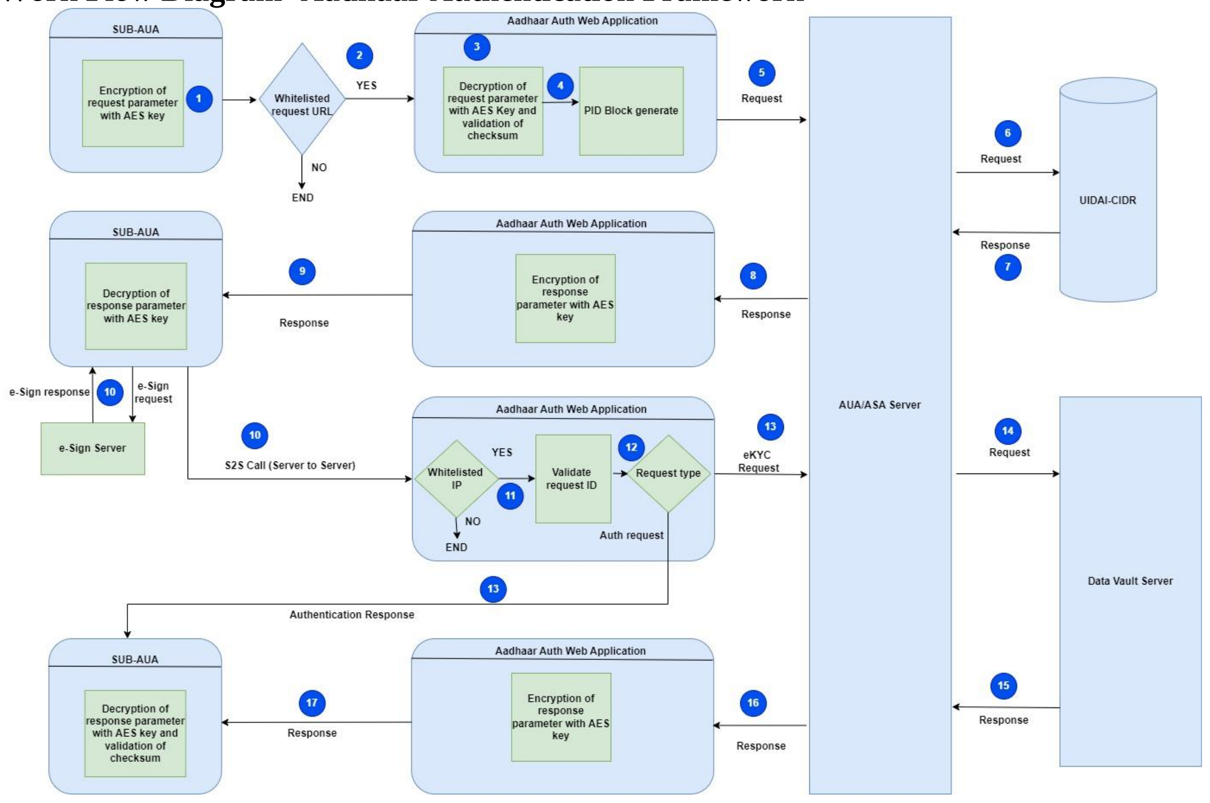
DOIT.C
Department of Information Technology & Communication, Rajasthan

RAJ E-SIGN PROJECT,
DEPARTMENT OF INFORMATION TECHNOLOGY & COMMUNICATION, GOR

RPP e-Sign API 1.1 INTEGRATION DOCUMENT (HTTP REDIRECTION MODE)

Introduction:- Presently, these Aadhaar authentication services are being provided through REST APIs. Now, AUA-DoIT&C is swiching from API model to HTTP redirection based request-response system. This document describes the workflow and steps for integration with Aadhaar Authentication Application of AUA-DoIT&C using HTTP redirection and to perform eSign using eSign API v1.1.

Work Flow Diagram- Aadhaar Authentication Framework


SUB-AUA
Encryption of request parameter with AES key
1
Whitelisted request URL
2
YES
NO
END
Aadhaar Auth Web Application
3
Decryption of request parameter with AES Key and validation of checksum
4
PID Block generate
5
Request
AUA/ASA Server
6
Request
UIDAI-CIDR
Response
7
8
Response
Encryption of response parameter with AES key
9
Response
Decryption of response parameter with AES key
e-Sign response
10
e-Sign request
e-Sign Server
S2S Call (Server to Server)
Whitelisted IP
11
Validate request ID
12
Request type
13
eKYC Request
Auth request
Authentication Response
14
Request
Data Vault Server
15
Response
16
Response
17
Response
Decryption of response parameter with AES key and validation of checksum

Transaction initiation

This step defines the method to create the Aadhaar authentication POST request from the Web application of the client. Client is required to post the request parameters on *Request URL*.

Request
UAT URL: **https://aadhaarauthtest.rajasthan.gov.in/AadhaarAuth**
Production URL: **https://aadhaarauth.rajasthan.gov.in/AadhaarAuth**
Method: POST

Client application needs to post only two parameters named "AppCode" and "Data", where "AppCode" is in plain string and "Data" is the encrypted JSON string containing the Auth request details.

```
<form action="<AuthRequestEncrypted>" method="POST">

<input type="hidden" name="AppCode" value="<AppCode>" />

<input type="hidden" name="Data" value="<Data>" />

<button type="submit">Submit</button>

</form>
```

- **Steps to create Data:**
- **Step 1:** Create the JSON string with request parameters as below

```
{
"SSOToken": "<SSOToken >",
"AppCode": "<AppCode >",
"RequestId": "<RequestId >",
 "AadhaarNo": "<AadhaarNo >",
 "RequestType": "<RequestType >",
 "ReturnURL": "<ReturnURL>",
 "Checksum": "<Checksum >",
 "VerificationType": "<VerificationType >",
  "OperatorIdType": "<OperatorIdType >",
  "OperatorId": "<OperatorId >",
  "Purpose": "<Purpose >",
}
```

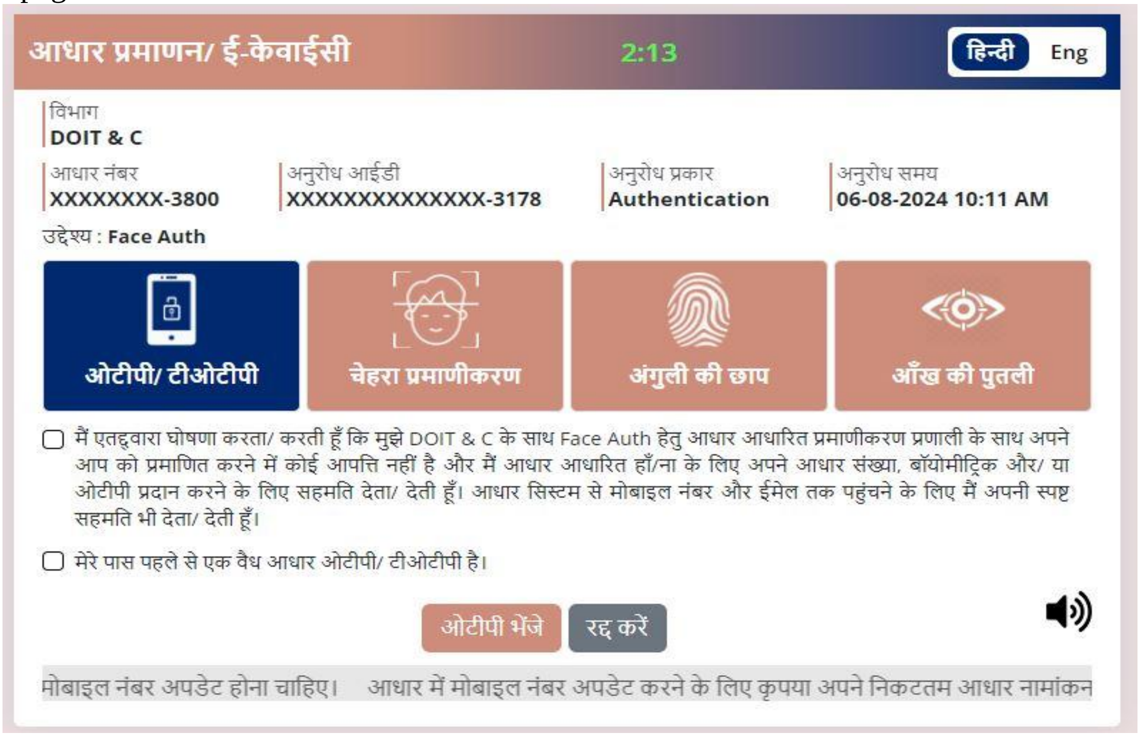
- **Step 2:** Encrypt the JSON string created in step 1 using AES key.

SR. No.	Request Parameter	Parameter Type	Parameter Constraints	Description
1	SSOToken	string		SSO session Token for SSO session validation

2	AppCode	string	Required Max Length: 32	Application-ID Issued by DoIT&C.
3	RequestId	String	Required Length: 18	This is the request id that is used to identify the particular request, it will be a unique ID and client will generate the request id. RequestId format should be as below: - *Sub-AUA Code + 8-digit Random number*. For Ex.- PRXXXXXXXX + 12345678
4	AadhaarNo	String	Required Min Length: 12 Max Length: 15	12 Digit Aadhaar Number/15 Digit Aadhaar Reference No./16 digit VID of beneficiary whose Aadhaar Face Authentication is to be performed. Encrypted using AES key.
5	RequestType	Int	Required	This parameter defines type of response required by the client application after face authentication. It accepts only two integer values- 1 - for Yes/No response 2 - for eKYC 3 – for eSign
6	ReturnURL	String	Required	It contains the return URL of the client application. It will be used for return to Client application after Face Authentication. This needs to be whitelisted at Aadhaar Authentication application.
7	Checksum	Hash		fixed checksum hash of values AppCode, RequestId, AadhaarNo, RequestType
8	VerificationType	Int	Required for Face Authentication	Verification Type parameter defines user type who will perform Face Authentication. It accepts only two integer values as below: - 1 - Beneficiary 2 - Authentication *Operator*
9	OperatorIdType	Int	Required if Face Authentication is to be performed through an Authentication Operator	This defines the ID Type of Authentication operator who will perform Face Authentication in his mobile. This field accepts only three integer values- **OperatorIdType** | **OperatorId** 1 | SSO ID 2 | 12 Digit Aadhaar Number 3 | 15 Digit Aadhaar Reference No.
10	OperatorId	String	Required if Face Authentication is to be performed through a Authentication Operator	This is the id of Authentication operator who will perform Face Authentication in his mobile. This field accepts only three string values based on OperatorIdType as below – **OperatorIdType** | **OperatorId**

				1 SSO ID 2 12 Digit Aadhaar Number 3 15 Digit Aadhaar Reference No.
11	Purpose	String	Required Max length: 100	This parameter defines the purpose of the Aadhaar Authentication Authentication i.e. name of any schemes or any other purpose.

After Creation of the POST request, client application will be redirected to Aadhaar Authentication application page.



विभाग
DOIT & C

आधार नंबर	अनुरोध आईडी	अनुरोध प्रकार	अनुरोध समय
XXXXXXXX-3800	XXXXXXXXXXXXXX-3178	Authentication	06-08-2024 10:11 AM

उद्देश्य : Face Auth

☐ मैं एतद्द्वारा घोषणा करता/ करती हूँ कि मुझे DOIT & C के साथ Face Auth हेतु आधार आधारित प्रमाणीकरण प्रणाली के साथ अपने आप को प्रमाणित करने में कोई आपत्ति नहीं है और मैं आधार आधारित हाँ/ना के लिए अपने आधार संख्या, बॉयोमीट्रिक और/ या ओटीपी प्रदान करने के लिए सहमति देता/ देती हूँ। आधार सिस्टम से मोबाइल नंबर और ईमेल तक पहुंचने के लिए मैं अपनी स्पष्ट सहमति भी देता/ देती हूँ।

☐ मेरे पास पहले से एक वैध आधार ओटीपी/ टीओटीपी है।

मोबाइल नंबर अपडेट होना चाहिए। आधार में मोबाइल नंबर अपडेट करने के लिए कृपया अपने निकटतम आधार नामांकन

- Aadhaar holder needs to select the authentication mode i.e. OTP, Fingerprint, Iris, PI/PA and face Auth etc. and perform authentication as per requirement.

Transaction Response

After execution of Aadhaar Authentication request, response is provided via POST method on the *ReturnURL* in JSON format. Response JSON will contain one parameters only “Data”, where “Data” is the encrypted JSON string containing the Aadhaar Auth response details.

Method: POST
Content-Type: application/json
Response json:

```
{
"Data": "",
}
```

“Data” parameter contains following JSON string in encrypted format i.e. Sub-AUA application will get following JSON after decryption of “Data” at client end:-

```
{
"RequestId": "RequestId",
"Message": "Your request has been executed successfully",
"ResponseCode": “200”
 }
```

Server to Server API call for encrypt text (eSign API 1.1)

Request

UAT URL:

https://apitest.sewadwaar.rajasthan.gov.in/app/live/RajeSign/Prod/createkey/webresources/generic/encrypttxt/service?client_id=

PRODUCTION URL:

https://api.sewadwaar.rajasthan.gov.in/app/live/Rajevault/Prod/Text/encrypt?client_id=

Method: POST

Content-Type: application/json

Request json: Request from e-Sign

```
{
 "transactionid":" Request ID generated for Aadhaar authentication ",
 "filecontant":"xxxx"
 }
```

Response

Method: POST

Content-Type: application/json

Response json:

Successful Response json:

```
{
    "Status": "1",
    "ResponseCode": "MS-111",
    "file": "xxxx",
    "ErrorMessage": "NA"
}
```

Failed Response json:

```
{
    "Status": "0",
    "ResponseCode": "xxxx",
    "ErrorMessage": "xxxx",
    "Transaction ID": " Request ID generated for Aadhaar authentication "
}
```

Details of request and response Parameters for e-Sign

SR. No.	Request/Response Parameter	Parameter Type	Description
1	file	String	Base64 of signed pay file
2	File	String	Base64 of user certificate
3	transactionid	String	Request ID generated for Aadhaar authentication.
4	filecontant	String	Base64 of pay file
5	Transaction ID	String	Request ID generated for Aadhaar authentication.
6	Status	String	0/1
7	ErrorMessage	String	Error Message related to e-sign application
8	TransactionId	String	Request ID generated for Aadhaar authentication
9	ResponseCode	String	Response code related to e-sign application

Sample Code for encryption and decryption:-

ASP .Net

---Encryption---

```
public static EncryptionResult Encrypt(string plainText, byte[] key)
{
EncriptionResult model = new();
try
{
using Aes aesAlg = Aes.Create();
aesAlg.Key = key;
aesAlg.GenerateIV();

ICryptoTransform encryptor = aesAlg.CreateEncryptor(aesAlg.Key, aesAlg.IV);
using MemoryStream msEncrypt = new();
msEncrypt.Write(aesAlg.IV, 0, aesAlg.IV.Length);

using (CryptoStream csEncrypt = new(msEncrypt, encryptor, CryptoStreamMode.Write))
{
using StreamWriter swEncrypt = new(csEncrypt);
swEncrypt.Write(plainText);
}
model.IsSuccess = true;
model.EncriptedData = msEncrypt.ToArray();
return model;
}
catch (Exception ex)
{
model.IsSuccess = false;
model.Message = ex.Message;
return model;
}
}
```

---**Decryption**---

```
private static EncryptionResult DecryptWithKey(byte[] cipherText, string keyString)
{
EncriptionResult model = new();
byte[] key = Encoding.UTF8.GetBytes(keyString);
try
{
using Aes aesAlg = Aes.Create();

aesAlg.Key = key;
byte[] iv = new byte[aesAlg.IV.Length];
Array.Copy(cipherText, 0, iv, 0, iv.Length);
aesAlg.IV = iv;
ICryptoTransform decryptor = aesAlg.CreateDecryptor(aesAlg.Key, aesAlg.IV);
using MemoryStream msDecrypt = new(cipherText, iv.Length, cipherText.Length - iv.Length);
using CryptoStream csDecrypt = new(msDecrypt, decryptor, CryptoStreamMode.Read);
using StreamReader srDecrypt = new(csDecrypt);
model.IsSuccess = true;
model.DecriptedData = srDecrypt.ReadToEnd();
}
catch (Exception ex)
{
model.IsSuccess = false;
model.Message = ex.Message;
}
return model;
}
```

--------------------------------------Class--

```
public class EncryptionResult{
public bool IsSuccess { get; set; }
public byte[] EncriptedData { get; set; } = new byte[0];
public string DecriptedData { get; set; } = string.Empty;
public string Message { get; set; } = string.Empty;
}
```

--------------------------------------Method---

```
 byte[] encData = Convert.FromBase64String('JSON STRING');
var requestData = Utilities.Decrypt(encData, Key);
```

--Generate Checksum--

```
 public static string ComputeSHA256Hash(string input)
{
using SHA256 sha256 = SHA256.Create();
byte[] inputBytes = Encoding.UTF8.GetBytes(input);
byte[] hashBytes = sha256.ComputeHash(inputBytes);
StringBuilder hex = new(hashBytes.Length * 2);
foreach (byte b in hashBytes)
{
hex.AppendFormat("{0:x2}", b);
}
return hex.ToString();
}
```

Java

--**Encryption**--

```
 public static byte[] encrypt(String plainText, byte[] key) throws Exception {
if (plainText == null || key == null)
{
throw new IllegalArgumentException("Plain text or key cannot be null");
}

//Create AES cipher
Cipher aesCipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
//Create secret key
SecretKeySpec secretKeySpec = new SecretKeySpec(key, "AES");
//Generate a random IV (Initialization Vector)

byte[] iv = new byte[16];
SecureRandom secureRandom = new SecureRandom();
secureRandom.nextBytes(iv);
IvParameterSpec ivParameterSpec = new IvParameterSpec(iv);


// Initialize cipher for encryption
aesCipher.init(Cipher.ENCRYPT_MODE, secretKeySpec, ivParameterSpec);
//Perform encryption
byte[] encrypted;
try (ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream()) {
//Write IV to the output stream
byteArrayOutputStream.write(iv);

//Encrypt plaintext
byte[] encryptedData = aesCipher.doFinal(plainText.getBytes(StandardCharsets.UTF_8));
byteArrayOutputStream.write(encryptedData);
encrypted = byteArrayOutputStream.toByteArray();
}
catch (IOException e) {
                    throw new RuntimeException("Error encrypting data", e);
          }

          return encrypted;
}
```

--Decryption--

```
public static String decrypt(byte[] encryptedData, byte[] key)
throws Exception {if (encryptedData == null || key == null) {
throw new IllegalArgumentException("Encrypted data or key cannot be null");
}
//Create AES cipher

Cipher aesCipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
//Extract IV from the encrypted data
byte[] iv = Arrays.copyOfRange(encryptedData, 0,
16); IvParameterSpec ivParameterSpec = new
IvParameterSpec(iv);
//Create secret key
SecretKeySpec secretKeySpec = new SecretKeySpec(key, "AES");
//Initialize cipher for decryption
aesCipher.init(Cipher.DECRYPT_MODE, secretKeySpec, ivParameterSpec);
//Perform decryption
byte[] decryptedData =
aesCipher.doFinal(Arrays.copyOfRange(encryptedData, 16,
encryptedData.length));
return new String(decryptedData, StandardCharsets.UTF_8);
}
```

------------Encryption calling method-----------------

```
String enKey = AppConfig.AES_key;
byte[] key = enKey.getBytes(StandardCharsets.UTF_8); // Convert key to byte array
//Encrypt JSON string
byte[] data = Aes256Encryption.encrypt(body.toString(), key);
//Encode encrypted data to Base64
String base64 = Base64.getEncoder().encodeToString(data);
```

------------Decryption calling method-----------------

```
 byte[] key = enKey.getBytes(StandardCharsets.UTF_8);
//Decrypt the data
String data = Aes256Encryption.decrypt(Base64.getDecoder().decode(encryptedData), key);
```

------------------------------**Checksum**------------------------------

```
public static String
generateChecksum(String input) {try {
//Create a SHA-256 MessageDigest instance
MessageDigest digest = MessageDigest.getInstance("SHA-256");
//Convert input string to bytes
byte[] inputBytes = input.getBytes("UTF-8");
//Compute the hash
byte[] hashBytes = digest.digest(inputBytes);

//Convert hash bytes to hexadecimal string
StringBuilder hex = new
StringBuilder(hashBytes.length * 2);for (byte b
: hashBytes) {
hex.append(String.format("%02x", b));
}
return hex.toString();
}
catch (NoSuchAlgorithmException |
java.io.UnsupportedEncodingException e) {throw new
RuntimeException("Error computing hash", e);
        }
}
```

===================== END OF DOCUMENT =====================