



RajCOMP Info Services Ltd.

(A Government of Rajasthan undertaking)

email: info.risl@rajasthan.gov.in

website: www.risl.rajasthan.gov.in

CIN : U72200RJ2010SGC033185

Ref. No. : F4.3(434)/RISL/Tech/2021

Date: 07-2025

Head of department,
All Departments,
Govt. of Rajasthan

Subject:- Regarding adoption of upgraded Jan Aadhaar APIs.

Sir/Madam,

With reference to the subject above, RISL has developed a new framework for the Jan Aadhaar API to enhance performance, strengthen security, and improve scalability, in line with the latest technological advancements.

Currently, various Jan Aadhaar APIs are being used across multiple departmental applications for resident authentication and direct benefit transfer services. With the upgraded Jan Aadhaar application and services now ready for use, it is essential that these new APIs are integrated into existing applications to ensure seamless and timely transition.

Timely migration to the new APIs will help prevent any potential service disruptions, as the existing Jan Aadhaar APIs will be discontinued effective **August 31, 2025**. Training programs about the upgraded APIs have already been conducted by the Rajasthan Jan Aadhaar Authority for the stakeholder departments on 16.07.2025 and 17.07.2025.

For any issues or clarifications, following officers may be contacted:

1. Sh. Suryakant Gupta, ACP (Dy. Director), DoIT&C, Mobile No.-9414870555
2. Sh. Sunil Choudhary, ACP (Dy. Director), DoIT&C, Mobile No.-9875005622

Your active cooperation is requested in implementing the updated APIs to ensure continuity and improvement in public service delivery.

Enclosed: Jan Aadhaar APIs Document.

Archana Singh
(Secretary, IT&C &
MD RISL)

Ref. No. : F4.3(434)/RISL/Tech/2021

Date: -07-2025

Copy the following for information and necessary action:-

1. Director (Technical), RISL.
2. Director, Rajasthan Jan Aadhaar Authority

Signature valid

Dheeraj Gaur

Digitally signed by Archana Singh
(Additional Secretary)
Designation : Chairman and
Managing Director
Date: 2025.07.21 11:09:10 IST
Reason: Approved





New Jan Aadhaar API Document

July-2025 (V 1.4)

RajComp Info Services Limited
1st Floor Yojana Bhawan, Tilak Marg,
C-Scheme, Jaipur- 302005

Contents

1. Overview:.....	2
2. Pre-requisites for Departments.....	3
3. API Understanding Steps	3
4. Technical Implementation.....	4
4.1 Jan Aadhaar Fetch Details API.....	4
4.2 DBT Cash Transaction PUSH API	10
4.3 DBT Non-Cash Transaction PUSH API	12
4.4 Entitlement Seeding API.....	14
4.5 Jan Aadhaar PI AUTH API.....	15
5. Parameters Description.....	17
6. Response Codes	18
7. Sample Code:	19
5.1 Java	19
5.2 .NET	21

1. Overview:

This document describes a secure, three-step API flow for fetching sensitive member details using a hybrid encryption approach and OTP-based authentication. The process follows a sign → encrypt → decrypt → verify model to ensure data confidentiality, integrity, and authenticity.

2. Pre-requisites for Departments

- a. **Identify Parameters:** Departments must identify the parameters required for fetching data from the Jan Aadhaar system.
- b. **Authentication Mode:** Departments need to select authenticate through Jan Aadhaar OTP.
- c. **Digital Signature:** Departments must procure the digital signature of the nodal authority.
- d. **Encryption Certificate:** Department should have public and private encryption certificate. The department will share its public key with Jan Aadhaar for encryption purposes and will use private key for decryption.
- e. **IP Address whitelisting** – Provide IP Addresses of the server to whitelist to access Jan Aadhaar API's.
- g. **Need to Raise On boarding Request:** User need to fill out the on boarding request form. After the request is approved by the Jan Aadhaar Authority, the requesting department will receive an email at the nodal officer's email ID containing a password-protected file, which contains the **Public Key, App Code and Scheme Code** for Jan Aadhaar integration. If department have Aadhaar Id or Member Id they need to directly call **Step-2** for Generate-OTP and Validate-OTP then get data.

3. API Understanding Steps

Step to be followed:

- **Step 1: Sign the Request**
 - Create a JSON object and digitally sign it using your **private RSA key**.
 - Append the signature in the "**signature**" field.
- **Step 2: Encrypt the Payload**
 - Encrypt the **entire JSON including the signature** using AES (symmetric key).
 - Encrypt the AES key using the Jan Aadhaar's **RSA public key**.

- Request will be sent in following format

ivBase64 : encryptedDataBase64 : encryptedAESKey

- **Step 3: Wrap the Payload**

the encrypted data in a wrapper:

```
{  
  "data": "<Encrypted_JSON_String (ivBase64:encryptedData:  
encryptedAESKey)>"  
}
```

- **Step 4: Response**

- a. The department will receive an encrypted and signed response. The response is encrypted using an AES key, and the AES key is encrypted using the department's public key. The response will be sent in the following format:

ivBase64 : encryptedDataBase64 : encryptedAESKey

- b. Decrypt the encrypted AES key using department's Private Key then decrypt the actual request using AES key.

4. Technical Implementation

4.1 Jan Aadhaar Fetch Details API

- **Step 1: Fetch Member List**

Endpoint

Method - POST

Raj Sewa Dwaar Service Name: JanAadhaar Fetch Member List

Service URL:

https://apitest.sewadwaar.rajasthan.gov.in/app/live/apiservice/janAadhaar/v1/member-list?client_id=

Request Format (Encrypted)

Request Format (Encrypted)

```
{
  "data": "<AES_Encrypted_Complete_JSON>"
}
```

Decrypted Payload

```
{
  "data": {
    "appCode": "APP001",
    "schemShortCode": "SCM001",
    "transactionId": "TXN123456",
    "janId": "123456789012"
  },
  "signature": "<RSA_Signature_Base64>"
}
```

Response (Encrypted)

```
{
  "data": "<Encrypted_Response_Data>"
}
```

Decrypted Response

```
{
  "response": {
    "status": true,
    "message": "Members fetched successfully",
    "responseCode": "JAN_200",
    "transactionId": "TXN123456",
    "schemeCode": "SCM001",
    "appCode": "APP001",
    "JanId": "123456789012",
    "data": [
      {
        "NAME_EN": "A*i* K*m*r",
        "MEMBERID": "MEM123456"
      },
      {

```

```

        "NAME_EN": "R*t* K*m*r*",
        "MEMBERID": "MEM789012"
    }
]
},
"signature": "<Base64_RSA_Signature_of_response>"
}

```

Error Response (Decrypted)

```

{
  "response": {
    "status": false,
    "message": "No data found for the given Enrollment ID or Jan
Aadhaar ID.",
    "responseCode": "JAN_404",
    "transactionId": "TXN123456",
    "data": null
  },
  "signature": "<RSA_Signature>"
}

```

Note: Parameters description is mentioned in Section 5 of the document.

- **Step 2: Generate OTP**

Endpoint

Method: POST

Raj Sewa Dwaar Service Name: JanAadhaar Generate OTP

Service URL:

https://apitest.sewadwaar.rajasthan.gov.in/app/live/apiservice/janAadhaar/v1/generate-otp?client_id=

Request Format (Encrypted)

```
{  
  
  "data": "<AES_Encrypted_Complete_JSON>"  
  
}
```

Request Format (Decrypted before encryption)

```
{  
  
  "data": {  
  
    "appCode": "APP001",  
  
    "schemShortCode": "SCM001",  
  
    "transactionId": "TXN987654",  
  
    "memberId": "MEM123456"  
  
  },  
  
  "signature": "<RSA_Signature>"  
  
}
```

Response Format (Encrypted)

```
{  
  
  "data": "<AES_Encrypted_Complete_JSON>"  
  
}
```

```
{  
  
  "response": {  
  
    "status": true,
```



```
"message": "OTP sent to xxxxxx1234 successfully",

"responseCode": "JAN_200",

"transactionId": "TXN987654",

"schemeCode": "SCM001",

"appCode": "APP001",

"tid": "TID20240625123456",

"JanId": null,

"data": null

},

"signature": "<RSA_Signature>"

}
```

Note: In case if OTP is not received to the beneficiary within 30 seconds, then same API can be called to resend OTP.

Note: Parameters description is mentioned in Section 5 of the document.

- **Step 3: Validate OTP & Fetch Requested Data**

Endpoint

Method: POST

**Raj Sewa Dwaar Service Name: JanAadhaar Validate OTP AND
Fetch Requested Data**

Service URL:

https://apitest.sewadwaar.rajasthan.gov.in/app/live/apiservice/janAadhaar/v1/validate-otp?client_id=

Request Format (Encrypted)

```
{
  "data": "<Encrypted_Complete_JSON>"
}
```

Request Format (Decrypted before encryption)

```
{
  "data": {
    "appCode": "APP001",
    "schemShortCode": "SCM001",
    "transactionId": "TXN000123",
    "memberId": "12345354225355",
    "tid": "TID20240625123456",
    "otp": "123456"
  },
  "signature": "<RSA_Signature>"
}
```

Response (Decrypted)

```
{
  "response": {
    "status": true,
    "message": "OTP validated successfully",
    "responseCode": "JAN_200",
    "transactionId": "TXN000123",
    "schemeCode": "SCM001",
    "appCode": "APP001",
    "tid": "TID20240625123456",
    "JanId": "123456789012",
    "data": {
      "NAME_LL": "अमित कुमार",
      "NAME_EN": "Amit Kumar",
      "DOB": "01-01-1990",

```

```

        "GENDER": "M"

        // Additional fields selected during onboarding
    }

    },

    "signature": "<RSA_Signature>"

}

```

Note: Parameters description is mentioned in Section 5 of the document.

4.2 DBT Cash Transaction PUSH API

Endpoint

Method - POST

Raj Sewa Dwaar Service Name: JanAadhaar DBT Cash Transaction PUSH
Service URL:

https://apitest.sewadwaar.rajasthan.gov.in/app/live/apiservice/janAadhaar/v1/pushCashTransation?client_id=

Request Format (Encrypted)

```

{
  "data": "<Encrypted_Complete_JSON>"
}

```

Request Format (Decrypted before encryption)

```

{
  "data": {
    "transactionId": "Unique Transaction Id",
    "schemeCode": "SCM001",
    "entitlementId": "12345",
    "entitlementMemId": "147856",
    "janaadhaarId": "5118431636",
    "janaadhaarMemId": "85362187957",
    "transactionMode": "AD/AC",
    "aadharNo": "444620016146065",

```

```
"bankAccNo": "6589654125",
"ifsc": "SBIN00001",
"micr": "",
"paymentAmount": "2000.00",
"paymentDate": "21-06-2024"
},
"signature": "<RSA_Signature>"
}
```

Response Format (Encrypted)

```
{
  "data": "<AES_Encrypted_Complete_JSON>"
}
```

Response Decrypted:

Success Response

```
{
  "response": {
    "status": true,
    "message": "Transaction successful",
    "responseCode": "JAN_200",
    "transactionId": "TXN000123",
    "schemeCode": "SCM001"
  },
  "signature": "<RSA_Signature>"
}
```

Failure Response

```
{
  "response": {
    "status": false,
    "message": "Duplicate Transaction ID",

```

```

    "responseCode": "JAN_501",
    "transactionId": "TXN000123",
    "schemeCode": "SCM001"
  },
  "signature": "<RSA_Signature>"
}

```

***Note:** Parameters description is mentioned in Section 5 of the document.*

4.3 DBT Non-Cash Transaction PUSH API

Endpoint

Method - POST

Raj Sewa Dwaar Service Name: JanAadhaar DBT Non Cash Transaction PUSH

Service URL:

https://apitest.sewadwaar.rajasthan.gov.in/app/live/apiservice/janAadhaar/v1/pushNonCashTransation?client_id=

Request Format (Encrypted):

```

{
  "data": "<Encrypted_Complete_JSON>"
}

```

Request Format (Decrypted before encryption):

```

{
  "data": {
    "transactionId": "Unique Transaction Id",
    "schemeCode": "SCM001",
    "entitlementId": "12345",
    "entitlementMemId": "147856",
    "janaadhaarId": "5118431636",
    "janaadhaarMemId": "85362187957",
    "items": [

```

```
{
  "quantity": "12",
  "commodity": "RICE",
  "unit": "KG"
},
{
  "aadharNo": "444620016146065"
},
{
  "signature": "<RSA_Signature>"
}
```

Response Format (Encrypted):

```
{
  "data": "<AES_Encrypted_Complete_JSON>"
}
```

Response Decrypted:

Success Response

```
{
  "response": {
    "status": true,
    "message": "Transaction successful",
    "responseCode": "JAN_200",
    "transactionId": "TXN000123",
    "schemeCode": "SCM001"
  },
  "signature": "<RSA_Signature>"
}
```

Failure Response

```
{
  "response": {
    "status": false,
    "message": "Duplicate Transaction",
    "responseCode": "JAN_501",
    "transactionId": "TXN000123",
    "schemeCode": "SCM001"
  },
  "signature": "<RSA_Signature>"
}
```

***Note:** Parameters description is mentioned in Section 5 of the document.*

4.4 Entitlement Seeding API

Endpoint

Method - POST

Raj Sewa Dwaar Service Name: JanAadhaar Entitlement Seeding

Service URL:

https://apitest.sewadwaar.rajasthan.gov.in/app/live/apiservice/janAadhaar/v1/entitlement-seeding?client_id=

Request Format (Encrypted):

```
{
  "data": "<Encrypted_Complete_JSON>"
}
```

Request Format (Decrypted before encryption):

```
{
  "data": {
    "janaadhaarId": "",
    "certValue": "",

```

```

        "janMemId": "",
        "docType": "",
        "message": "",
        "expDate": "",
        "certNo": "",
        "name": "",
        "documentId": "",
        "issueDate": "",
        "issuingAuth": "",
        "status": ""
    },
    "signature": "<RSA_Signature>"
}

```

Response Format (Encrypted):

```

{
  "data": "<AES_Encrypted_Complete_JSON>"
}

```

Response Decrypted:

```

{
  "response": {
    "status": true,
    "message": "Transaction successful",
    "responseCode": "JAN_200",
    "transactionId": "TXN000123",
    "schemeCode": "SCM001"
  },
  "signature": "<RSA_Signature>"
}

```

Note: Parameters description is mentioned in Section 5 of the document.

4.5 Jan Aadhaar PI AUTH API

Endpoint

Method - POST

Raj Sewa Dwaar Service Name: JanAadhaar PI AUTH

Service URL:

https://apitest.sewadwaar.rajasthan.gov.in/app/live/apiservice/janAadhaar/v1/janaadhaar-piauth?client_id=

Request Format (Encrypted):

```
{  
  "data": "<Encrypted_Complete_JSON>"  
}
```

Request Format (Decrypted before encryption):

```
{  
  "data": {  
    "schemeCode": "SCM001",  
    "janId": "Enrollment/Jan Aadhaar Id/ Jan Member Id",  
    "matchValue": [  
      {  
        "aadhaarRefId": "Aadhaar Ref. Id",  
        "nameEng": "Name English",  
        "gender": "Gender",  
        "dob": "(DD/MM/YYYY)",  
        "bankAccount": "Bank Account",  
        "ifsc": "IFSC code",  
        "fatherNameEng": "Father Name English",  
        "motherNameEng": "Mother Name English",  
        "spouseNameEng": "Spouse Name Eng",  
        "mobileNo": "Mobile No"  
      }  
    ]  
  },  
  "signature": "<RSA_Signature>"  
}
```

Note:

- A match on at least one parameter value is mandatory.
- Any number of values can be passed, for unused fields, pass an empty string ("").

Response Format (Encrypted):

```
{
  "data": "<AES_Encrypted_Complete_JSON>"
}
```

Response Decrypted:

```
{
  "response": {
    "status": true,
    "message": "Transaction successful",
    "responseCode": "JAN_200",
    "transactionId": "TXN000123",
    "schemeCode": "SCM001"
  },
  "signature": "<RSA_Signature>"
}
```

***Note:** Parameters description is mentioned in Section 5 of the document.*

5. Parameters Description

S. No.	Field Name	Field Type	Description
1	schemShortCode	String	Scheme Code provided by Jan Aadhaar during onboarding process
2	janId	String	10 digit Jan Aadhaar Id / 15 digit Enrollment Id
3	appCode	String	App Code provided by Jan Aadhaar during onboarding process
4	transactionId	String	Unique ID for tracking the transaction
5	memberId	Number	11 digit Jan Aadhaar Member Id
6	signature	String	Digital signature for request validation
7	tid	Number	Unique ID for OTP validation.
8	otp	Number	One-Time Password sent to the user for authentication
9	aadhaarRefId	Number	Reference Number generated from the Tokenize Service of UIDAI against Aadhaar Id

S. No.	Field Name	Field Type	Description
10	nameEng	String	Jan Aadhaar Member Name in English
11	gender	String	Gender of the individual (e.g., Male, Female, Transgender)
12	dob	String	Date of birth (format: DD-MM-YYYY)
13	bankAccount	Number	Bank account number
14	ifsc	String	IFSC code of the bank branch
15	fatherNameEng	String	Father's name in English
16	motherNameEng	String	Mother's name in English
17	spouseNameEng	String	Spouse's name in English (if applicable)
18	mobileNo	Number	Mobile number
19	janaadhaarId	String	10 digit Jan Aadhaar Id / 15 digit Enrollment Id
20	certValue	String	Value of the certificate (likely Base64 or textual)
21	janMemId	Number	11 digit Jan Aadhaar Member Id
22	docType	String	Type of document (e.g., Birth Certificate, Bonafied Certificate, Caste Certificate etc.)
23	message	String	Message or note against the response status
24	expDate	String	Expiry date of the document (format: DD-MM-YYYY)
25	certNo	String	Certificate number
26	name	String	Member Name
27	documentId	String	Unique ID of the document provided by eVault
28	issueDate	String	Document issue date (format: DD-MM-YYYY)
29	issuingAuth	String	Name of the issuing authority
30	status	String	Status of the document or record (e.g., Active, Inactive)
31	quantity	Number	Quantity value (e.g., ration quantity)
32	commodity	String	Name/type of commodity
33	unit	Number	Unit of measurement
34	entitlementId	Number	Unique entitlement Id generated/provided by the Department
35	entitlementMemId	Number	Unique entitlement Member Id generated/provided by the Department
36	janaadhaarMemId	Number	11 digit Jan Aadhaar Member Id
37	aadharNo	Number	12 Digit Aadhaar Id
38	paymentAmount	Number	Transaction Amount in Rs.
39	paymentDate	String	Date of payment (format: DD-MM-YYYY)

6. Response Codes

S.No.	Error Codes	Description
1	JAN_000	Success

2	JAN_001	Error processing request, please try again!
3	JAN_002	Scheme is not onboarded.
4	JAN_003	IP address is not whitelisted
5	JAN_004	Scheme Short Code is mandatory.
6	JAN_005	Jan Aadhaar ID is mandatory.
7	JAN_006	Unsupported token
8	JAN_007	General exception occurred during token validation Token cannot be null or empty
9	JAN_008	Internal Server Error. Service Unavailable.
10	JAN_101	Aadhaar eKYC Process failed.
11	JAN_102	User Terminated eKYC Process after OTP Generation.
12	JAN_409	Invalid Token Format
13	JAN_410	Invalid token signature.
14	JAN_404	No data found for the given Enrolment ID or Jan Aadhaar ID
15	JAN_501	Duplicate Transaction ID.

7. Sample Code:

5.1 Java

1. Method to encrypt using AES

```

public String encryptDataWithAES(String data) throws Exception {
    SecretKey aesKey = generateAESKey();
    Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
    byte[] iv = new byte[16];
    new SecureRandom().nextBytes(iv);
    IvParameterSpec ivParameterSpec = new IvParameterSpec(iv);
    cipher.init(Cipher.ENCRYPT_MODE, aesKey, ivParameterSpec);
    byte[] encryptedData = cipher.doFinal(data.getBytes());
    // Base64 encode the encrypted data and IV
    String encryptedDataBase64 =
        Base64.getEncoder().encodeToString(encryptedData);
    String ivBase64 = Base64.getEncoder().encodeToString(iv);
    // Encrypt the AES key with RSA
    String encryptedAESKey = encryptAESKeyWithRSA(aesKey);
    // Combine the AES-encrypted data and IV for transmission
    return ivBase64 + ":" + encryptedDataBase64 + ":" + encryptedAESKey;
}

```

2. Method to Decrypt using AES

```
// Decrypt the data using AES key
public String decryptDataWithAES(String encryptedData) throws Exception {
    // Split the encrypted string into IV, encrypted data, and encrypted AES key
    String[] parts = encryptedData.split(":");
    String ivBase64 = parts[0];
    String encryptedDataBase64 = parts[1];
    String encryptedAESKey = parts[2];
    // Decode the base64-encoded IV, encrypted data, and AES key
    byte[] iv = Base64.getDecoder().decode(ivBase64);
    byte[] encryptedDataBytes = Base64.getDecoder().decode(encryptedDataBase64);
    // Decrypt the AES key using RSA
    SecretKey aesKey = decryptAESKeyWithRSA(encryptedAESKey);
    // Decrypt the data with AES
    IvParameterSpec ivParameterSpec = new IvParameterSpec(iv);
    Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
    cipher.init(Cipher.DECRYPT_MODE, aesKey, ivParameterSpec);
    byte[] decryptedData = cipher.doFinal(encryptedDataBytes);
    return new String(decryptedData);
}
```

3. Method to encrypt the AES key using public key

```
// Encrypt the AES key with RSA public key
public String encryptAESKeyWithRSA(SecretKey aesKey) throws Exception {
    Cipher cipher = Cipher.getInstance(RSA_ALGORITHM);
    cipher.init(Cipher.ENCRYPT_MODE, publicKey); //use JanAdhaar's public key
    byte[] encryptedAESKey = cipher.doFinal(aesKey.getEncoded());
    return Base64.getEncoder().encodeToString(encryptedAESKey);
}
```

4. Method to decrypt the AES key using public key

```
public SecretKey decryptAESKeyWithRSA(String encryptedAESKey) throws
Exception {
    byte[] encryptedAESKeyBytes = Base64.getDecoder().decode(encryptedAESKey);
    Cipher cipher = Cipher.getInstance(RSA_ALGORITHM);
    cipher.init(Cipher.DECRYPT_MODE, privateKey); // use your private key
    byte[] decryptedAESKeyBytes = cipher.doFinal(encryptedAESKeyBytes);
    return new SecretKeySpec(decryptedAESKeyBytes, AES_ALGORITHM);
}
```

5. Key Generator

```
private SecretKey generateAESKey() throws Exception {
    KeyGenerator keyGen = KeyGenerator.getInstance(AES_ALGORITHM);
```

```
keyGen.init(256); // 256-bit AES key
return keyGen.generateKey();
}
```

6. Data Signing

```
public String signData(String data) throws Exception {
    // Sign data with the private key
    Signature signature = Signature.getInstance("SHA256withRSA");
    signature.initSign(privateKey); //use your private key
    signature.update(data.getBytes());
    byte[] digitalSignature = signature.sign();
    return Base64.getEncoder().encodeToString(digitalSignature);
}
```

5.2 .NET

1. Method to encrypt using AES

```
public class CryptoHelper
{
    private const string AES_ALGORITHM = "AES";
    private const string RSA_ALGORITHM = "RSA";
    private RSA publicKey; // Initialize with your public key
    private RSA privateKey; // Initialize with your private key
    public string EncryptDataWithAES(string data)
    {
        // Generate a new AES key
        using (var aesKey = GenerateAESKey())
        {
            // Encrypt the data using AES
            using (var cipher = Aes.Create())
            {
                cipher.Mode = CipherMode.CBC;
                cipher.Padding = PaddingMode.PKCS7;

                byte[] iv = new byte[16]; // AES block size is 16 bytes
                using (var rng = new RNGCryptoServiceProvider())
                {
                    rng.GetBytes(iv);
                }
                cipher.IV = iv;
                cipher.Key = aesKey.Key;
            }
        }
    }
}
```

```

        using (var encryptor = cipher.CreateEncryptor())
        {
            byte[] encryptedData =
encryptor.TransformFinalBlock(Encoding.UTF8.GetBytes(data), 0,
data.Length);

            // Base64 encode the encrypted data and IV
            string encryptedDataBase64 =
Convert.ToBase64String(encryptedData);
            string ivBase64 = Convert.ToBase64String(iv);

            // Encrypt the AES key with RSA
            string encryptedAESKey = EncryptAESKeyWithRSA(aesKey);

            // Combine the AES-encrypted data and IV for transmission
            return
$"{{ivBase64}}:{{encryptedDataBase64}}:{{encryptedAESKey}}";
        }
    }
}

```

2. [Method to Decrypt using AES](#)

```

public string DecryptDataWithAES(string encryptedData)
{
    // Split the encrypted string into IV, encrypted data, and encrypted
AES key
    string[] parts = encryptedData.Split(':');
    string ivBase64 = parts[0];
    string encryptedDataBase64 = parts[1];
    string encryptedAESKey = parts[2];

    // Decode the base64-encoded IV, encrypted data, and AES key
    byte[] iv = Convert.FromBase64String(ivBase64);
    byte[] encryptedDataBytes =
Convert.FromBase64String(encryptedDataBase64);

    // Decrypt the AES key using RSA
    using (var aesKey = DecryptAESKeyWithRSA(encryptedAESKey))
    {
        // Decrypt the data with AES
    }
}

```

```

        using (var cipher = Aes.Create())
        {
            cipher.Mode = CipherMode.CBC;
            cipher.Padding = PaddingMode.PKCS7;
            cipher.IV = iv;
            cipher.Key = aesKey.Key;

            using (var decryptor = cipher.CreateDecryptor())
            {
                byte[] decryptedData =
                decryptor.TransformFinalBlock(encryptedDataBytes, 0,
                encryptedDataBytes.Length);
                return Encoding.UTF8.GetString(decryptedData);
            }
        }
    }
}

```

3. Method to encrypt the AES key using public key

```

public string EncryptAESKeyWithRSA(Aes aesKey)
{
    using (var cipher = RSA.Create())
    {
        cipher.ImportParameters(publicKey.ExportParameters(false));
        byte[] encryptedAESKey = cipher.Encrypt(aesKey.Key,
        RSAEncryptionPadding.Pkcs1);
        return Convert.ToBase64String(encryptedAESKey);
    }
}

```

4. Method to decrypt the AES key using public key

```

public Aes DecryptAESKeyWithRSA(string encryptedAESKey)
{
    byte[] encryptedAESKeyBytes =
    Convert.FromBase64String(encryptedAESKey);
    using (var cipher = RSA.Create())
    {
        cipher.ImportParameters(privateKey.ExportParameters(true));
        byte[] decryptedAESKeyBytes =
        cipher.Decrypt(encryptedAESKeyBytes, RSAEncryptionPadding.Pkcs1);
        var aes = Aes.Create();
        aes.Key = decryptedAESKeyBytes;
        return aes;
    }
}

```



```
}
```

```
}
```

5. [Key Generator](#)

```
private Aes GenerateAESKey()
```

```
{
```

```
    var aes = Aes.Create();
```

```
    aes.KeySize = 256; // 256-bit AES key
```

```
    aes.GenerateKey();
```

```
    return aes;
```

```
}
```

6. [Data Signing](#)

```
public string SignData(string data)
```

```
{
```

```
    using (var signature = RSA.Create())
```

```
    {
```

```
        signature.ImportParameters(privateKey.ExportParameters(true));
```

```
        byte[] dataBytes = Encoding.UTF8.GetBytes(data);
```

```
        byte[] digitalSignature = signature.SignData(dataBytes,
```

```
HashAlgorithmName.SHA256, RSASignaturePadding.Pkcs1);
```

```
        return Convert.ToBase64String(digitalSignature);
```

```
    }
```

```
}}
```