

Encryption / Decryption Integration Guide



The NSE Clearing Limited (National Clearing)
Exchange Plaza, Plot No. C/1, G Block,
Bandra-Kurla Complex, Bandra (E),
Mumbai - 400 051

NSE Clearing Confidential

Notice

© Copyright NSE Clearing Ltd (NCL). All rights reserved. Unpublished rights reserved under applicable copyright and trades secret laws.

The contents, ideas and concepts presented herein are proprietary and confidential.
Duplication and disclosure to others in whole, or in part is prohibited

Table of Contents

AES Algorithm Integration Guide	3
1. Overview	3
2. Key & IV Requirements (Mandatory)	3
3. Encryption Flow.....	4
4. Decryption Flow	4
5. Sample code	4
RSA Algorithm Integration Guide.....	6
1. Overview	6
2. Purpose	6
3. Key Requirements	7
4. Data Format Expectations.....	7
5. RSA Decryption Flow (Members Side)	7
6. Reference Java Decryption Code	8
7. Members Responsibilities.....	8
Frequently Asked Questions (FAQs).....	9

AES Algorithm Integration Guide

Algorithm: AES-256-CBC with PKCS5 Padding

Encoding: UTF-8

Cipher Text Format: Base64

1. Overview

For secure message exchange between systems, we use **AES (Advanced Encryption Standard)** with the following configuration:

Parameter	Value
Algorithm	AES
Mode	CBC (Cipher Block Chaining)
Padding	PKCS5Padding
Secret Key Size	256 bits (32 bytes)
IV Size	128 bits (16 bytes)
Text Encoding	UTF-8
Cipher Output	Base64 encoded string

This document provides **reference implementation** and **rules** to ensure interoperability.

2. Key & IV Requirements (Mandatory)

Secret Key

- Must be **256 bits (32 bytes)**
- Must be the **same on both encryption and decryption sides**
- Should be generated securely and stored safely

Initialization Vector (IV)

- Must be **16 bytes (128 bits)**
- Must be shared securely with the Members
- Should be unique per session if possible (recommended)

Incorrect key or IV length will cause decryption failure

3. Encryption Flow

1. Convert plain text to UTF-8 bytes
2. Encrypt using AES/CBC/PKCS5Padding
3. Encode encrypted bytes to **Base64**
4. Send Base64 cipher text to the receiver

4. Decryption Flow

1. Decode **Base64** cipher text
2. Decrypt using the same AES key and IV
3. Convert decrypted bytes to UTF-8 string

5. Sample code

Constants

AES = "AES";

**AES_TRANSFORMATION = "AES/CBC/PKCS5Padding"; CHARSET_NAME =
"UTF-8";**

Sample code for encryption

```
public String encrypt(String      plainText, byte[] secretKeyBytes, byte[] ivBytes)
{
    try {
        if (plainText == null || secretKeyBytes == null || ivBytes == null) {
            return null;
        }
        SecretKeySpec keySpec = new SecretKeySpec(secretKeyBytes, AES);
        IvParameterSpec ivSpec = new IvParameterSpec(ivBytes);

        Cipher cipher = Cipher.getInstance(AES_TRANSFORMATION);    cipher.init(Cipher.ENCRYPT_MODE,
        keySpec, ivSpec);

        byte[] encryptedBytes = cipher.doFinal(plainText.getBytes(CHARSET_NAME));
        return Base64.getEncoder().encodeToString(encryptedBytes);

    } catch (Exception e)    {
        e.printStackTrace();
        return null;
    }
}
```

Sample code of decryption

```
public String decrypt(String base64CipherText, byte[] secretKeyBytes, byte[] ivBytes) {    try {  
    if (base64CipherText == null || secretKeyBytes == null || ivBytes == null) {  
return null;  
    }  
  
    SecretKeySpec keySpec = new SecretKeySpec(secretKeyBytes, AES);        IvParameterSpec  
ivSpec = new IvParameterSpec(ivBytes);  
  
    Cipher cipher = Cipher.getInstance(AES_TRANSFORMATION);        cipher.init(Cipher.DECRYPT_MODE,  
keySpec, ivSpec);  
    byte[] decryptedBytes =  
        cipher.doFinal(Base64.getDecoder().decode(base64CipherText));  
  
    return new String(decryptedBytes, CHARSET_NAME);  
  
    } catch (Exception e) {  
e.printStackTrace();        return  
null;  
    }  
}
```

RSA Algorithm Integration Guide

Algorithm: RSA

Transformation: RSA/ECB/PKCS1Padding

Encoding: UTF-8

Cipher Text Format: Base64

1. Overview

RSA (Rivest–Shamir–Adleman) is an **asymmetric encryption algorithm** commonly used to:

- Securely exchange symmetric keys (e.g., AES keys)
- Protect small sensitive payloads
- Establish trust between client and server

In our integration, RSA is used for **decryption on the server side** using a **private key**, while encryption is performed on the client side using the corresponding **public key**.

2. Purpose

RSA encryption is used to securely transfer sensitive information (such as encrypted messages or symmetric keys) from the client to the server.

- **NCL side:** Encrypts data using the **RSA Public Key**
- **Members side:** Decrypts data using the **RSA Private Key**

This document explains **how the NCL data is decrypted on our side** using RSA. **Transformation**

Explanation

- **RSA**
Rivest–Shamir–Adleman asymmetric encryption algorithm.
- **ECB**
Required placeholder in Java Cipher API for RSA.
RSA is not a block cipher, so ECB mode does not apply practically.
- **PKCS1Padding**
PKCS#1 v1.5 padding used for RSA encryption/decryption

3. Key Requirements

RSA Key Pair

Key Type	Usage
Public Key	Used by NCL to encrypt data
Private Key	Used by Membersss to decrypt data

Key Size

- Minimum recommended: **2048 bits**

4. Data Format Expectations

- Plain text is encoded using **UTF-8**
- Encrypted output is **Base64 encoded**
- The Base64 string is sent to the server for decryption

5. RSA Decryption Flow (Members Side)

- Receive Base64-encoded encrypted data from client
- Decode Base64 string into byte array
- Initialize RSA cipher using private key
- Decrypt encrypted bytes
- Convert decrypted bytes to UTF-8 string

6. Reference Java Decryption Code

```
private static final String RSA_TRANSFORMATION = "RSA/ECB/PKCS1Padding";

@Override
public String decrypt(String base64Encrypted, PrivateKey privateKey) {    try {
    // Initialize RSA cipher
    Cipher cipher = Cipher.getInstance(RSA_TRANSFORMATION);
    cipher.init(Cipher.DECRYPT_MODE, privateKey);

    // Decode Base64 and decrypt
    byte[] encryptedBytes = Base64.getDecoder().decode(base64Encrypted);
    byte[] decryptedBytes = cipher.doFinal(encryptedBytes);

    // Return decrypted string
    return new String(decryptedBytes, "UTF-8");

} catch (Exception e) {
    e.printStackTrace();
    return null;
}
}
```

7. Members Responsibilities

The client must ensure:

- Data is encrypted using **RSA public key**
- Same transformation: RSA/ECB/PKCS1Padding
- Plain text encoded using **UTF-8**
- Encrypted output sent as **Base64 string** Any mismatch will result in decryption failure.

NOTE:

Members must strictly follow the same algorithm, padding, encoding, and data format for successful integration for both the algorithms.

Frequently Asked Questions (FAQs)

1. What is the minimum key length?

- Minimum key length should be 2048 bits

2. What should be the subject/addtext Syntax?

- Use the following subject syntax while generating the certificate

C=<Country Code>

ST=<State>

L=<City>

O=<OrganizationName>

OU=<DepartmentName>

CN=<DomainOrApplicationName>

3. What should be the certificate format and extension?

- Certificate should be X.509 format. PEM encoded with '.pem' file extension

4. Whether self-signed certificate is acceptable?

- Recommend using CA signed certificate.

5. What should be the maximum expiry period?

- Expected expiry period will be one year.

*** End of Document ***