

## Profile 360 Writer

**[i]** SIT environment RSA public/private keys have been changed, please refer below link to get updated public keys.  
<https://code.airtelworld.in:7990/bitbucket/projects/APB/repos/customerprofile/browse/products-sync/keys?at=profile360>

The nature of the api is such that it can accept multiple customers data in it like shown in below curl. And the response of each customer will be separate. Example shown in api response section below.

SIT Url (POST) : <https://apbsit110-234.airtelbank.com/customer-products>

### PAYOUT ENCRYPTION STEPS:

1. Generate random uuid (this will be our key for this API particular API call)
2.
  - a. AES encryption of the data using key from step 1 new padding - **AES/GCM/NoPadding** and hashAlgo - **PBKDF2WithHmacSHA256**
  - b. URL encoding of encrypted data
  - c. RSA encrypt key using public key transformation **RSA/ECB/OAEPwithSHA-256andMGF1Padding**.
3. Put key in the header naming **key** with RSA encrypted, like shown in below curl. Url Encoding is not needed.
4. Put the payload data encrypted by AES algo in **nd-json** format i.e. new line delimited json. For example you need to push data for 2 customers,

then encrypt each customer json separately using AES, followed by URLEncoder. Similarly encrypt other customer json also with AES, followed by URLEncoder.

In this way you will have 2 separate lines in the payload, each for a different customer as shown in the curl below.

### API Curl below

```
1 curl --location --request POST
'https://10.56.110.234/customer-products' \
2 --header 'channel: IOS' \
3 --header 'clientId: ONBOARDING' \
4 --header 'correlationId: 234ww' \
5 --header 'key:
HSwqyaAkVqbpey2WzThajkKBVuX3erUCDY8ByUIs6a2083cJ
WqWJ6fa0mppR43Xa0XjnAyzMkoFmxwEWMVK0x16TtMFmfap/
anYtlalAeiEuorxN1tvx7L3RXz1qTb9dmgyIEE0isLyq4Jib
Ia2ifrVRyIYqXgxtf5ctmTx9D4tDPAYjEzLVAJaum5HaUVyx
+EH8wneJiCoiW44wpU0F00e6gm6kqd/ybdsGC/rFT0nbby/
LmUiia2bDkm5KmhAlKy1pPiAEc1NDejf5G89jE0KpEPd00/cP
oKSETMQ/FggwSBhYFKZJWHuE0TDRJff8+R/6zM4qiiA+m59R
BHtt2A==' \
6 --header 'Authorization: Basic
cHJvZHvjdHMtc3luYy1zaXQ6chJvZHvjdHMtc3luYy1zaXQ=
' \
7 --data-raw
'Eb%2BJFTMdLSfsdYIVX9NRPMJD1RFiq2mqL%2F5xsNhSkN0
1BHCakVZjMXMg2QnnUBoPCG85%2B1nd2SA0geT1Jgb7mJgat
gmJ9I7t2BzBC69K6Sg4j1u3W0XyafUfTFkzh1F9gAZcKXOMM
WWCJfhmW8yKA17S0NrvaZjBvA1TaKUmriU8jyzxK1FtGLCKeH
WFgyRC51uEjjSu63zs0nTAc6t%2B10RyI3TFIgrH7yuxJ2h9
jKsMIKAjiPAuNHZ6G6hAs%2F102jFHkCdHNZ7ALphQa%2Fa%
2Fks1qnYvN0Hje5mIUhn7v91VFFjNQGacLacit4RQ%3D%3D
8 Eb%2BJFTMdLSfsdYIVX9NRPMJD1RFiq2mqL%2F5xsNhSkN01
BHCakVZjMXMg2QnnUBoPCG85%2B1nd2SA0geT1Jgb7mJgatg
mJ9I7t2BzBC69K6Sg4j1u3W0XyafUfTFkzh1F9gAZcKXOMMW
WcJfhmW8yKA17S0NrvaZjBvA1TaKUmriU8jyzxK1FtGLCKeHW
FgyRC51uEjjSu63zs0nTAc6t%2B10RyI3TFIgrH7yuxJ2h9j
KsMIKAjiPAuNHZ6G6hAs%2F102jFHkCdHNZ7ALphQa%2Fa%
2Fks1qnYvN0Hje5mIUhn7v91VFFjNQGacLacit4RQ%3D%3D'
```

### Headers:

Key	Value
Content-Type	application/x-ndjson
channel	API
correlationId	{Any unique alphanumeric chars}
Authorization	Basic cHJvZHvjdHMtc3luYy1zaXQ6chJvZHvj

	dHMtc3luYy1zaXQ=
clientId	{will be shared for each team separately}
key	{Encryption key. Can be any unique alphanumeric} This key must be encrypted with RSA

# AES encryption Code

## Imports

```
1 import java.nio.charset.StandardCharsets;
2 import java.security.MessageDigest;
3 import java.security.SecureRandom;
4 import java.util.Arrays;
5 import java.util.Base64;
6
7 import javax.crypto.Cipher;
8 import javax.crypto.spec.GCMParameterSpec;
9 import javax.crypto.spec.SecretKeySpec;
10
11 import
12     org.springframework.boot.context.properties.ConfigurationProperties;
13 import
14     org.springframework.cloud.context.config.annotation.RefreshScope;
15 import
16     org.springframework.stereotype.Component;
17
18 import
19     com.airtelbank.updateprofile.exception.AesException;
20
21 import lombok.Getter;
22 import lombok.Setter;
23 import lombok.extern.slf4j.Slf4j;
```

AesUtil.java

```
1  @Slf4j
2  @Getter
3  @Setter
4  @Component
5  @ConfigurationProperties(prefix =
"application.aes")
6  public class AesUtil {
7
8      private String padding;
9      private String hashalgo;
10
11     private int blockSize = 128;
12     private int iterationCount = 20;
13     private byte[] salt = new byte[] { -88,
-101, -56, 50, 86, 52, -29, 3 };
14
15     public String decrypt(String passKey,
String encData, byte[] salt) {
16         try {
```

```

17         PBEKeySpec pbeKeySpec = new
18         PBEKeySpec(passKey.toCharArray(), salt,
19         iterationCount, blockSize);
20
21         SecretKeySpec keySpec = new
22         SecretKeySpec(SecretKeyFactory.getInstance(hash
23         algo).generateSecret(pbeKeySpec).getEncoded(),
24         "AES");
25
26         GCMParameterSpec gcmParameterSpec =
27         new GCMParameterSpec(blockSize, salt);
28
29         Cipher cipher =
30         Cipher.getInstance(padding);
31         cipher.init(Cipher.DECRYPT_MODE,
32         keySpec, gcmParameterSpec);
33
34         byte[] dec =
35         cipher.doFinal(Base64.getDecoder().decode(encDa
36         ta.getBytes())));
37
38         return new String(dec,
39         StandardCharsets.UTF_8);
40     } catch (Exception e) {
41         log.error("AesUtil | decrypt", e);
42     }
43     return StringUtils.EMPTY;
44 }
45
46     public String encrypt(String passKey,
47     String plnData, byte[] salt) {
48
49     try {
50
51         PBEKeySpec pbeKeySpec = new
52         PBEKeySpec(passKey.toCharArray(), salt,
53         iterationCount, blockSize);
54
55         SecretKeySpec keySpec = new
56         SecretKeySpec(SecretKeyFactory.getInstance(hash
57         algo).generateSecret(pbeKeySpec).getEncoded(),
58         "AES");
59
60         GCMParameterSpec gcmParameterSpec =
61         new GCMParameterSpec(blockSize, salt);
62
63         Cipher cipher =
64         Cipher.getInstance(padding);
65
66         cipher.init(Cipher.ENCRYPT_MODE,
67         keySpec, gcmParameterSpec);
68
69         byte[] enc =
70         cipher.doFinal(plnData.getBytes(StandardCharset
71         s.UTF_8));
72
73         return
74         URLEncoder.encode(Base64.getEncoder().encodeToS
75         tring(enc), StandardCharsets.UTF_8.name())
76     } catch (Exception e) {
77         log.error("AesUtil | encrypt", e);
78     }
79     return StringUtils.EMPTY;
80 }
81
82 }
83
84 }
```

## RSA encryption Code

### Imports

```
1 import java.io.FileInputStream;
2 import java.security.Key;
3 import java.security.KeyFactory;
4 import java.security.PrivateKey;
5 import java.security.PublicKey;
6 import java.security.spec.PKCS8EncodedKeySpec;
7 import java.security.spec.X509EncodedKeySpec;
8 import java.util.Base64;
9
10 import javax.crypto.Cipher;
11
12 import org.apache.commons.io.IOUtils;
13 import org.apache.commons.lang3.StringUtils;
14 import
15     org.springframework.boot.context.properties.Con
16     figurationProperties;
17 import
18     org.springframework.stereotype.Component;
19 import lombok.Getter;
20 import lombok.Setter;
21 import lombok.extern.slf4j.Slf4j;
22 import java.security.Security;
23 import java.nio.file.Files;
24 import java.nio.file.Paths
```

## RsaUtil.java

```
1 package
2     com.airtelbank.customerprofile.products.sync.a
3     pi.filter;
4
5 import java.io.IOException;
6 import java.nio.file.Files;
7 import java.nio.file.Paths;
8 import java.security.InvalidKeyException;
9 import java.security.KeyFactory;
10 import java.security.NoSuchAlgorithmException;
11 import java.security.PrivateKey;
12 import java.security.Provider;
13 import java.security.PublicKey;
14 import java.security.Security;
15 import
16     java.security.spec.InvalidKeySpecException;
17 import java.security.spec.PKCS8EncodedKeySpec;
18 import java.security.spec.X509EncodedKeySpec;
19 import java.util.Base64;
20
21 import javax.crypto.BadPaddingException;
22 import javax.crypto.Cipher;
23 import javax.crypto.IllegalBlockSizeException;
24 import javax.crypto.NoPaddingException;
25
26 import org.apache.commons.lang3.StringUtils;
27 import
28     org.springframework.boot.context.properties.Co
29     nfigurationProperties;
30 import
31     org.springframework.stereotype.Component;
```

```
26
27 import lombok.Getter;
28 import lombok.Setter;
29 import lombok.extern.slf4j.Slf4j;
30
31 @Slf4j
32 @Getter
33 @Setter
34 @Component
35 @ConfigurationProperties("application.rsa")
36 public class RsaUtil {
37
38     private String privateKeyPath;
39     private String publicKeyPath;
40
41
42     /**
43      * @author Anand Mukut Tirkey
44      * @param string to be encrypted
45      * @return encrypted string in base 64.
46      * empty string in case of issue
47      */
48     public String encrypt(String string,
49                           String hashAlgo) {
50         if(StringUtils.isBlank(string)) return
51             StringUtils.EMPTY;
52         Provider cp =
53             Security.getProvider("SunJCE");
54         Cipher cipher;
55         try {
56             cipher =
57                 Cipher.getInstance("RSA/ECB/OAEPwith" +
58                     hashAlgo + "andMGF1Padding", cp);
59             cipher.init(Cipher.ENCRYPT_MODE,
60                         getPublicKeyFromFile());
61             byte[] enc =
62                 cipher.doFinal(string.getBytes());
63             return
64                 Base64.getEncoder().encodeToString(enc);
65         } catch (NoSuchAlgorithmException |
66                  NoSuchPaddingException e) {
67             log.error("RsaEncDec | encrypt
68                  NoSuchAlgorithmException /
69                  NoSuchPaddingException", e);
70         } catch (InvalidKeyException e) {
71             log.error("RsaEncDec | encrypt
72                  InvalidKeyException", e);
73         } catch (InvalidKeySpecException e) {
74             log.error("RsaEncDec | encrypt
75                  InvalidKeySpecException", e);
76         } catch (IOException e) {
77             log.error("RsaEncDec | encrypt
78                  IOException", e);
79         } catch (IllegalBlockSizeException e)
80         {
81             log.error("RsaEncDec | encrypt
82                  IllegalBlockSizeException", e);
83         } catch (BadPaddingException e) {
84             log.error("RsaEncDec | encrypt
85                  BadPaddingException", e);
86         } catch (Exception e) {
87             log.error("RsaEncDec | encrypt
88                  Exception", e);
89         }
90     }
91
92     /**
93
```

```
75     * @author Anand Mukut Tirkey
76     * @param string to be decrypted in base
77     * @return decrypted string. empty string
78     * in case of issue
79     */
80     public String decrypt(String base64string,
81     String hashAlgo) {
82         if(StringUtils.isBlank(base64string))
83             return StringUtils.EMPTY;
84         Provider cp =
85             Security.getProvider("SunJCE");
86         Cipher cipher;
87         try {
88             cipher =
89                 Cipher.getInstance("RSA/ECB/OAEPwith" +
90                     hashAlgo + "andMGF1Padding", cp);
91             cipher.init(Cipher.DECRYPT_MODE,
92             getPrivateKeyFromFile());
93             byte[] dec =
94                 cipher.doFinal(Base64.getDecoder().decode(base
95                 64string));
96             return new String(dec);
97         } catch (BadPaddingException e) {
98             log.error("RsaEncDec | decrypt
99             BadPaddingException", e);
100        } catch (NoSuchAlgorithmException |
101            NoSuchPaddingException e) {
102            log.error("RsaEncDec | decrypt
103            NoSuchAlgorithmException /
104            NoSuchPaddingException", e);
105        } catch (IllegalBlockSizeException e)
106        {
107            log.error("RsaEncDec | decrypt
108            IllegalBlockSizeException", e);
109        } catch (InvalidKeyException e) {
110            log.error("RsaEncDec | decrypt
111            InvalidKeyException", e);
112        } catch (InvalidKeySpecException e) {
113            log.error("RsaEncDec | decrypt
114            InvalidKeySpecException", e);
115        } catch (IOException e) {
116            log.error("RsaEncDec | decrypt
117            IOException", e);
118        } catch (Exception e) {
119            log.error("RsaEncDec | decrypt
120            Exception", e);
121        }
122        return StringUtils.EMPTY;
123    }
124
125    private PublicKey getPublicKeyFromFile()
126    throws IOException, NoSuchAlgorithmException,
127    InvalidKeySpecException {
128        byte[] keyBytes =
129            Files.readAllBytes(Paths.get(publicKeyPath));
130        X509EncodedKeySpec spec = new
131            X509EncodedKeySpec(keyBytes);
132        KeyFactory kf =
133            KeyFactory.getInstance("RSA");
134        return kf.generatePublic(spec);
135    }
136
137    private PrivateKey getPrivateKeyFromFile()
138    throws IOException, NoSuchAlgorithmException,
139    InvalidKeySpecException {
```

```

115         byte[] keyBytes =
116         Files.readAllBytes(Paths.get(privateKeyPath));
117         PKCS8EncodedKeySpec spec = new
118         PKCS8EncodedKeySpec(keyBytes);
119         KeyFactory kf =
120         KeyFactory.getInstance("RSA");
121         return kf.generatePrivate(spec);
122     }

```

### Plain Request Json looks like below

```

1  [
2   {
3     "custId": 69,
4     "mobileNumber": "98765432100",
5     "accountNumber": 9876543210,
6     "products": [
7       {
8         "id": "PRAN",
9         "type": "FD",
10        "status": "Active",
11        "deliveryStatus": "pending",
12        "issuanceDate": "2025-02-03",
13        "expiryDate": "2026-02-03"
14      },
15      {
16        "id": "8989586904",
17        "type": "AUTO_LOAD",
18        "status": "Active",
19        "deliveryStatus": "pending",
20        "issuanceDate": "2025-02-03",
21        "expiryDate": "2026-02-03"
22      },
23      {
24        "id": "8989586904",
25        "type": "SWEEP",
26        "status": "Active",
27        "deliveryStatus": "pending",
28        "issuanceDate": "2025-02-03",
29        "expiryDate": "2026-02-03"
30      },
31      {
32        "id": "8989586904",
33        "type": "SWEEP",
34        "status": "Active",
35        "deliveryStatus": "pending",
36        "issuanceDate": "2025-02-03",
37        "expiryDate": "2026-02-03"
38      },
39      {
40        "id": "8989586904",
41        "type": "DIGI_GOLD",
42        "status": "Active",
43        "deliveryStatus": "pending",
44        "issuanceDate": "2025-02-03",
45        "expiryDate": "2026-02-03",
46        "customField1": "",
47        "customField2": ""

```

```

48         "customField3": "",  

49         "customField4": "",  

50         "customField5": ""  

51     }  

52   ]  

53 },  

54 {  

55   "custId": 70,  

56   "natlId": "98765432100",  

57   "accountNumber": 9876543210,  

58   "products": [  

59     {  

60       "id": "PRAN",  

61       "type": "FD",  

62       "status": "Active",  

63       "deliveryStatus": "pending",  

64       "issuanceDate": "2025-02-03",  

65       "expiryDate": "2026-02-03"  

66     },  

67     {  

68       "id": "8989586904",  

69       "type": "AUTO_LOAD",  

70       "status": "Active",  

71       "deliveryStatus": "pending",  

72       "issuanceDate": "2025-02-03",  

73       "expiryDate": "2026-02-03"  

74     },  

75     {  

76       "id": "8989586904",  

77       "type": "SWEEP",  

78       "status": "Active",  

79       "deliveryStatus": "pending",  

80       "issuanceDate": "2025-02-03",  

81       "expiryDate": "2026-02-03"  

82     },  

83     {  

84       "id": "8989586904",  

85       "type": "SWEEP",  

86       "status": "Active",  

87       "deliveryStatus": "pending",  

88       "issuanceDate": "2025-02-03",  

89       "expiryDate": "2026-02-03"  

90     },  

91     {  

92       "id": "8989586904",  

93       "type": "DIGI_GOLD",  

94       "status": "Active",  

95       "deliveryStatus": "pending",  

96       "issuanceDate": "2025-02-03",  

97       "expiryDate": "2026-02-03",  

98       "customField1": "",  

99       "customField2": "",  

100      "customField3": "",  

101      "customField4": "",  

102      "customField5": ""  

103    }  

104  ]  

105}  

106 ]

```

**Request Pojo (To be used in java code)**

```

1  @Setter
2  @Getter
3  @ToString
4  @Builder
5  @NoArgsConstructor
6  @AllArgsConstructor
7  @JsonInclude(JsonInclude.Include.NON_EMPTY)
8  @JsonIgnoreProperties(ignoreUnknown = true)
9  public class Customer {
10
11      @NotNull(message = "is Required")
12      @Positive(message = "Must be a number")
13      @Max(value=99999999999L, message = "Must
14      not be greater than 15 digits")
15      private Long custId;
16
17      @NotEmpty(message = "NatId Cant be null or
empty")
18      @Pattern(regexp = "^\\d{10,11}$", message =
"must be exactly 10 or 11 digits")
19      private String mobileNumber;
20
21      @NotNull(message = "is Required")
22      @Positive(message = "Must be a number")
23      @Min(value = 1000000000L, message = "Must
be exactly 10 digits")
24      @Max(value = 9999999999L, message = "Must
be exactly 10 digits")
25      private Long accountNumber;
26
27      @NotEmpty(message = "Products list cannot
be empty")
28      private List<Map<String, Object>> products;
29
30  }

```

### Payload Mandatory fields

Key	Data type	Size
custId	Long	15
mobileNumbe r	String	11
accountNumb er	Long	10
products[0].id	String (This product/feature id)	50
products[0].ty pe	String (Possible values below)	50
	1 DIGI_GOLD, 2 FD, 3 NCMC, 4 DEBIT_CARD,	

	5 SURAKSHA, 6 REWARD123, 7 REWARDMINI, 8 TRAVELLER, 9 REWARD123PLUS, 10 LOAN, 11 DBT, 12 FASTAG, 13 INSURANCE, 14 APY, 15 SWEEP, 16 AUTO_LOAD, 17 REKYC_DUE_DATE, 18 UPI_ENABLED, 19 SMART_PHONE, 20 SEGMENTATION, 21 URBAN, 22 RURAL	
--	--	--

### Payload non mandatory fields

Key	Data type	Size (characters)
products[0].status	String	10
products[0].deliveryStatus	String	10
products[0].issuanceDate	String	YYYY-MM-dd
products[0].expiryDate	String	YYYY-MM-dd
products[0].customerTomField1 products[0].customerTomField2 products[0].customerTomField3 products[0].customerTomField4 products[0].customerTomField5	String	60
	<p><span style="color: blue;">i</span> Just in case of extra data needs to be inserted. It can be inserted in custom fields provided           </p>	

that the refresh rate of the field should not be high.

## Api Response:

```
1  {
2      "meta": {
3          "code": "1000",
4          "description": "Success",
5          "status": 0
6      },
7      "data": [
8          {
9              "custId": 69,
10             "status": 0,
11             "message": "Insertion Success"
12         },
13         {
14             "custId": 70,
15             "status": 1,
16             "message": "Insertion Failed. Will
retry"
17         },
18         {
19             "custId": 71,
20             "status": 0,
21             "message": "Partial Success. Failed
to insert in cache. We Will try to sync in
some time"
22         }
23     ]
24 }
```

- 💡 There are 3 types of responses that the api gives with respect to each customer.

1. **Insertion Success** - means insertion of customer data is successfully done in db and cache both.
2. **Insertion Failed. Will retry**- means insertion of customer data failed completely.

Neither data is fed in db nor in cache, but will retry to feed data in both db and cache.

Use case team can also retry after some time.

**3. Partial Success.** Failed to insert in cache.

We Will try to sync in some time - means data is fed in db but failed to persist in cache. The cache sync will be automatically done in sometime or client can retry hitting the api once again.

**4. status = 0 can be considered as successful insertion whereas status = 1, can be considered as insertion failure and use case teams can retry in this case.**

### Response Meta Codes:

Status	Code	Description
0	1000	Success
1	1004	Data is invalid. Please check.
1	1005	Generic Error
1	1007	Request headers are missing
1	1009	Data parse error

### Want to Sync one time data to kafka

Kafka channel insertion is not approved by the management. If you want to push history data you can call the writer api asynchronously. Or you can

use a kafka channel at your end and consume the api.

If there is huge history data for back filling you can use kafka channel but only for this purpose.

**kafka cluster** - “10.56.21.247:9092”

**kafka topic** - “banking-product-products-sync-topic”

**Data format:** Only one customer data per message is allowed in kafka **unlike** writer api where multiple customers' data can be sent. Data format is shown below.

**Message key:** Message key must be present. Pass cust\_id as message key in string format.

```
1  {
2      "custId": 69,
3      "mobileNumber": "98765432100",
4      "accountNumber": 9876543210,
5      "products": [
6          {
7              "id": "PRAN",
8              "type": "FD",
9              "status": "Active",
10             "deliveryStatus": "pending",
11             "issuanceDate": "2025-02-03",
12             "expiryDate": "2026-02-03"
13         },
14         {
15             "id": "8989586904",
16             "type": "AUTO_LOAD",
17             "status": "Active",
18             "deliveryStatus": "pending",
19             "issuanceDate": "2025-02-03",
20             "expiryDate": "2026-02-03"
21         },
22         {
23             "id": "8989586904",
24             "type": "SWEEP",
25             "status": "Active",
26             "deliveryStatus": "pending",
27             "issuanceDate": "2025-02-03",
28             "expiryDate": "2026-02-03"
29         },
30         {
31             "id": "8989586904",
32             "type": "SWEEP",
33             "status": "Active",
34             "deliveryStatus": "pending",
35             "issuanceDate": "2025-02-03",
36             "expiryDate": "2026-02-03"
37         },
38         {
39             "id": "8989586904",
40             "type": "DIGI_GOLD",
41             "status": "Active",
42             "deliveryStatus": "pending",
43             "issuanceDate": "2025-02-03",
44             "expiryDate": "2026-02-03",
```

```
45     "customField1": "",  
46     "customField2": "",  
47     "customField3": "",  
48     "customField4": "",  
49     "customField5": ""  
50   }  
51 ]  
52 }
```



