


Profile 360 Writer

 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>

PAYLOAD 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/
  anYtla1AeiEuorxN1tvx7L3RXz1qTb9dmgyIEE0isLyq4Jib
  Ia2ifrVRyIYqXgxtf5ctmTx9D4tDPAYjEzLVAJaum5HaUVyx
  +EH8wneJiCoiW44wpU0F00e6gm6kqd/ybdsGC/rFT0nbbby/
  LmUia2bDkm5KmhAlKy1pPiAEc1NDejf5689jE0KpEPd00/cP
  oKSETMQ/FggwSBhYFKZJWHuE0TDRJff8+R/6zM4qiiA+m59R
  Bhtt2A==' \
6 --header 'Authorization: Basic
  cHJvZHVjdHMtc3luYy1zaXQ6cHJvZHVjdHMtc3luYy1zaXQ=
  ' \
7 --data-raw
  'Eb%2BJFTMdLSfsdYIVX9NRPMJD1RFiq2mqL%2F5xsNhSkN0
  1BHCAkVZjMXMg2QnnUBoPCG85%2B1nd2SA0geT1Jgb7mJgatg
  gmJ9I7t2BzBC69K6Sg4jlu3W0XyafUfTFkzh1F9gAZcKX0MM
  WWcJfhmW8yKA17S0NrvAzjBvAlTaKUmU8jyzxK1FtGLCKeH
  WFgyRC51uEJjSu63zs0nTAc6t%2B10RyI3TFIgrH7yuxJ2h9
  jKsMIKAjiPAnUHZ6G6hAs%2F102jFHKCdHNZ7ALphQa%2Fa%
  2Fks1qnYvN0Hje5mIUhn7v91VFFjNQGacLacit4RQ%3D%3D
8 Eb%2BJFTMdLSfsdYIVX9NRPMJD1RFiq2mqL%2F5xsNhSkN01
  BHCAkVZjMXMg2QnnUBoPCG85%2B1nd2SA0geT1Jgb7mJgatg
  mJ9I7t2BzBC69K6Sg4jlu3W0XyafUfTFkzh1F9gAZcKX0MMW
  WcJfhmW8yKA17S0NrvAzjBvAlTaKUmU8jyzxK1FtGLCKeHW
  FgyRC51uEJjSu63zs0nTAc6t%2B10RyI3TFIgrH7yuxJ2h9j
  KsMIKAjiPAnUHZ6G6hAs%2F102jFHKCdHNZ7ALphQa%2Fa%2
  Fks1qnYvN0Hje5mIUhn7v91VFFjNQGacLacit4RQ%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
    org.springframework.boot.context.properties.Con
    figurationProperties;
12 import
    org.springframework.cloud.context.config.annota
    tion.RefreshScope;
13 import
    org.springframework.stereotype.Component;
14
15 import
    com.airtelbank.updateprofile.exception.AesExcep
    tion;
16
17 import lombok.Getter;
18 import lombok.Setter;
19 import lombok.extern.slf4j.Slf4j;
20

```

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
PBEKeySpec(passKey.toCharArray(), salt,
iterationCount, blockSize);
18         SecretKeySpec keySpec = new
SecretKeySpec(SecretKeyFactory.getInstance(hash
algo).generateSecret(pbeKeySpec).getEncoded(),
"AES");
19         GCMParameterSpec gcmParameterSpec =
new GCMParameterSpec(blockSize, salt);
20
21         Cipher cipher =
Cipher.getInstance(padding);
22         cipher.init(Cipher.DECRYPT_MODE,
keySpec, gcmParameterSpec);
23
24         byte[] dec =
cipher.doFinal(Base64.getDecoder().decode(encDa
ta.getBytes()));
25         return new String(dec,
StandardCharsets.UTF_8);
26     } catch (Exception e) {
27         log.error("AesUtil | decrypt", e);
28     }
29     return StringUtils.EMPTY;
30 }
31
32     public String encrypt(String passKey,
String plnData, byte[] salt) {
33         try {
34             PBEKeySpec pbeKeySpec = new
PBEKeySpec(passKey.toCharArray(), salt,
iterationCount, blockSize);
35             SecretKeySpec keySpec = new
SecretKeySpec(SecretKeyFactory.getInstance(hash
algo).generateSecret(pbeKeySpec).getEncoded(),
"AES");
36             GCMParameterSpec gcmParameterSpec =
new GCMParameterSpec(blockSize, salt);
37
38             Cipher cipher =
Cipher.getInstance(padding);
39             cipher.init(Cipher.ENCRYPT_MODE,
keySpec, gcmParameterSpec);
40
41             byte[] enc =
cipher.doFinal(plnData.getBytes(StandardCharset
s.UTF_8));
42             return
URLLEncoder.encode(Base64.getEncoder().encodeToS
tring(enc), StandardCharsets.UTF_8.name())
43         } catch (Exception e) {
44             log.error("AesUtil | encrypt", e);
45         }
46         return StringUtils.EMPTY;
47     }
48 }
49 }
50

```

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
    org.springframework.boot.context.properties.Con
    figurationProperties;
15 import
    org.springframework.stereotype.Component;
16
17 import lombok.Getter;
18 import lombok.Setter;
19 import lombok.extern.slf4j.Slf4j;
20 import java.security.Security;
21 import java.nio.file.Files;
22 import java.nio.file.Paths

```

RsaUtil.java

```

1 package
    com.airtelbank.customerprofile.products.sync.a
    pi.filter;
2
3 import java.io.IOException;
4 import java.nio.file.Files;
5 import java.nio.file.Paths;
6 import java.security.InvalidKeyException;
7 import java.security.KeyFactory;
8 import java.security.NoSuchAlgorithmException;
9 import java.security.PrivateKey;
10 import java.security.Provider;
11 import java.security.PublicKey;
12 import java.security.Security;
13 import
    java.security.spec.InvalidKeySpecException;
14 import java.security.spec.PKCS8EncodedKeySpec;
15 import java.security.spec.X509EncodedKeySpec;
16 import java.util.Base64;
17
18 import javax.crypto.BadPaddingException;
19 import javax.crypto.Cipher;
20 import javax.crypto.IllegalBlockSizeException;
21 import javax.crypto.NoSuchPaddingException;
22
23 import org.apache.commons.lang3.StringUtils;
24 import
    org.springframework.boot.context.properties.Co
    nfigurationProperties;
25 import
    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         return StringUtils.EMPTY;
91     }
92
93     /**

```

```

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

```

```

115     byte[] keyBytes =
Files.readAllBytes(Paths.get(privateKeyPath));
116     PKCS8EncodedKeySpec spec = new
PKCS8EncodedKeySpec(keyBytes);
117     KeyFactory kf =
KeyFactory.getInstance("RSA");
118     return kf.generatePrivate(spec);
119 }
120
121 }
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     "natId": "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=999999999999L, message = "Must
not be greater than 15 digits")
14      private Long custId;
15
16      @NotEmpty(message = "NatId Cant be null or
empty")
17      @Pattern(regexp = "^\\d{10,11}$", message =
"must be exactly 10 or 11 digits")
18      private String mobileNumber;
19
20      @NotNull(message = "is Required")
21      @Positive(message = "Must be a number")
22      @Min(value = 1000000000L, message = "Must
be exactly 10 digits")
23      @Max(value = 9999999999L, message = "Must
be exactly 10 digits")
24      private Long accountNumber;
25
26      @NotEmpty(message = "Products list cannot
be empty")
27      private List<Map<String, Object>> products;
28
29  }
30


```

Paylod Mandatory fields

Key	Data type	Size
custId	Long	15
mobileNumber	String	11
accountNumber	Long	10
products[0].id	String (This is product/feature id)	50
products[0].type	String (Possible values below) <div> 1 DIGI_GOLD, 2 FD, 3 NCMC, 4 DEBIT_CARD, </div>	50

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].customField1 products[0].customField2 products[0].customField3 products[0].customField4 products[0].customField5	String <div> Just in case of extra data needs to be inserted. It can be inserted in custom fields provided</div>	60

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 }  
25
```

 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	Cod e	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 }
```