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

```
curl --location --request POST
'https://10.56.110.234/customer-products' \
--header 'channel: IOS' \
--header 'clientId: ONBOARDING' \
--header 'correlationId: 234ww' \
--header 'key:
HSwqyaAkVqbpey2WzThajkKBVuX3erUCDY8ByUIs6a2083cJ
WqWJ6faOmppR43Xa0XjnAyzMkoFmxwEWMVK0x16TtMFMfap/
anYtlalAeiEuorxN1tvx7L3RXzlqTb9dmgyIEEOisLyq4Jib
Ia2ifrVRyIYqXgxtf5ctmTx9D4tDPAYjEzLVAJaum5HaUVyx
+EH8wneJiCoiW44wpUOF00e6gm6kqd/ybdsGC/rFTOnbbby/
LmUia2bDkm5KmhAlKy1pPiAEclNDejf5G89jE0KpEPdO0/cP
oKSETMQ/FggwSBhYFKZJWHuE0TDRJff8+R/6zM4qiiA+m59R
BHtt2A==' \
--header 'Authorization: Basic
cHJvZHVjdHMtc3luYy1zaXQ6cHJvZHVjdHMtc3luYy1zaXQ=
' \
--data-raw
'Eb%2BJFTMdLSfsdYIVX9NRPMJD1RFiq2mqL%2F5xsNhSkNO
lBHCAkVZjMXMg2QnnUBoPCG85%2B1nd2SA0geT1Jgb7mJgat
gmJ9I7t2BzBC69K6Sg4jlu3W0XyafUfTFkzhlF9gAZcKXOMM
WWcJfhmW8yKA17S0NrvAzjBvAlTaKUmrU8jyzxKlFtGLCKeH
WFgyRC51uEJjSu63zs0nTAc6t%2B10RyI3TFIgrH7yuxJ2h9
jKsMIKAjiPAnUHZ6G6hAs%2F102jFHKCdHNZ7ALphQa%2Fa%
2Fks1qnYvN0Hje5mIUhn7v91VFFjNQGacLacit4RQ%3D%3D
Eb%2BJFTMdLSfsdYIVX9NRPMJD1RFiq2mqL%2F5xsNhSkNOl
BHCAkVZjMXMg2QnnUBoPCG85%2B1nd2SA0geT1Jgb7mJgatg
mJ9I7t2BzBC69K6Sg4jlu3W0XyafUfTFkzhlF9gAZcKXOMMW
WcJfhmW8yKA17S0NrvAzjBvAlTaKUmrU8jyzxKlFtGLCKeHW
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

```java
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.security.SecureRandom;
import java.util.Arrays;
import java.util.Base64;

import javax.crypto.Cipher;
import javax.crypto.spec.GCMParameterSpec;
import javax.crypto.spec.SecretKeySpec;

import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.cloud.context.config.annotation.RefreshScope;
import org.springframework.stereotype.Component;

import com.airtelbank.updateprofile.exception.AesException;

import lombok.Getter;
import lombok.Setter;
import lombok.extern.slf4j.Slf4j;

```

AesUtil.java

```java
@Slf4j
@Getter
@Setter
@Component
@ConfigurationProperties(prefix = "application.aes")
public class AesUtil {

    private String padding;
    private String hashalgo;

    private int blockSize = 128;
    private int iterationCount = 20;
    private byte[] salt = new byte[] { -88, -101, -56, 50, 86, 52, -29, 3 };

    public String decrypt(String passKey, String encData, byte[] salt) {
        try {
            PBEKeySpec pbeKeySpec = new PBEKeySpec(passKey.toCharArray(), salt, iterationCount, blockSize);
            SecretKeySpec keySpec = new SecretKeySpec(SecretKeyFactory.getInstance(hashalgo).generateSecret(pbeKeySpec).getEncoded(), "AES");
            GCMParameterSpec gcmParameterSpec = new GCMParameterSpec(blockSize, salt);

            Cipher cipher = Cipher.getInstance(padding);
            cipher.init(Cipher.DECRYPT_MODE, keySpec, gcmParameterSpec);

            byte[] dec = cipher.doFinal(Base64.getDecoder().decode(encData.getBytes()));
            return new String(dec, StandardCharsets.UTF_8);
        } catch (Exception e) {
            log.error("AesUtil | decrypt", e);
        }
        return StringUtils.EMPTY;
    }

    public String encrypt(String passKey, String plnData, byte[] salt) {
        try {
            PBEKeySpec pbeKeySpec = new PBEKeySpec(passKey.toCharArray(), salt, iterationCount, blockSize);
            SecretKeySpec keySpec = new SecretKeySpec(SecretKeyFactory.getInstance(hashalgo).generateSecret(pbeKeySpec).getEncoded(), "AES");
            GCMParameterSpec gcmParameterSpec = new GCMParameterSpec(blockSize, salt);

            Cipher cipher = Cipher.getInstance(padding);
            cipher.init(Cipher.ENCRYPT_MODE, keySpec, gcmParameterSpec);

            byte[] enc = cipher.doFinal(plnData.getBytes(StandardCharsets.UTF_8));
            return URLEncoder.encode(Base64.getEncoder().encodeToString(enc), StandardCharsets.UTF_8.name())
        } catch (Exception e) {
            log.error("AesUtil | encrypt", e);
        }
        return StringUtils.EMPTY;
    }

}

```

RSA encryption Code

Imports

```java
import java.io.FileInputStream;
import java.security.Key;
import java.security.KeyFactory;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.spec.PKCS8EncodedKeySpec;
import java.security.spec.X509EncodedKeySpec;
import java.util.Base64;

import javax.crypto.Cipher;

import org.apache.commons.io.IOUtils;
import org.apache.commons.lang3.StringUtils;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.stereotype.Component;

import lombok.Getter;
import lombok.Setter;
import lombok.extern.slf4j.Slf4j;
import java.security.Security;
import java.nio.file.Files;
import java.nio.file.Paths
```

RsaUtil.java

```java
package com.airtelbank.customerprofile.products.sync.api.filter;

import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.security.InvalidKeyException;
import java.security.KeyFactory;
import java.security.NoSuchAlgorithmException;
import java.security.PrivateKey;
import java.security.Provider;
import java.security.PublicKey;
import java.security.Security;
import java.security.spec.InvalidKeySpecException;
import java.security.spec.PKCS8EncodedKeySpec;
import java.security.spec.X509EncodedKeySpec;
import java.util.Base64;

import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.NoSuchPaddingException;

import org.apache.commons.lang3.StringUtils;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.stereotype.Component;

import lombok.Getter;
import lombok.Setter;
import lombok.extern.slf4j.Slf4j;

@Slf4j
@Getter
@Setter
@Component
@ConfigurationProperties("application.rsa")
public class RsaUtil {

    private String privateKeyPath;
    private String publicKeyPath;


    /**
     * @author Anand Mukut Tirkey
     * @param string to be encrypted
     * @return encrypted string in base 64. empty string in case of issue
     * */
    public String encrypt(String string, String hashAlgo) {
        if(StringUtils.isBlank(string)) return StringUtils.EMPTY;
        Provider cp = Security.getProvider("SunJCE");
        Cipher cipher;
        try {
            cipher = Cipher.getInstance("RSA/ECB/OAEPwith" + hashAlgo + "andMGF1Padding", cp);
            cipher.init(Cipher.ENCRYPT_MODE, getPublicKeyFromFile());
            byte[] enc = cipher.doFinal(string.getBytes());
            return Base64.getEncoder().encodeToString(enc);
        } catch (NoSuchAlgorithmException | NoSuchPaddingException e) {
            log.error("RsaEncDec | encrypt NoSuchAlgorithmException / NoSuchPaddingException", e);
        } catch (InvalidKeyException e) {
            log.error("RsaEncDec | encrypt InvalidKeyException", e);
        } catch (InvalidKeySpecException e) {
            log.error("RsaEncDec | encrypt InvalidKeySpecException", e);
        } catch (IOException e) {
            log.error("RsaEncDec | encrypt IOException", e);
        } catch (IllegalBlockSizeException e) {
            log.error("RsaEncDec | encrypt IllegalBlockSizeException", e);
        } catch (BadPaddingException e) {
            log.error("RsaEncDec | encrypt BadPaddingException", e);
        } catch (Exception e) {
            log.error("RsaEncDec | encrypt Exception", e);
        }
        return StringUtils.EMPTY;
    }

    /**
```

```java
     * @author Anand Mukut Tirkey
     * @param string to be decrypted in base 64
     * @return decrypted string. empty string in case of issue
     * */
    public String decrypt(String base64string, String hashAlgo) {
        if(StringUtils.isBlank(base64string)) return StringUtils.EMPTY;
        Provider cp = Security.getProvider("SunJCE");
        Cipher cipher;
        try {
            cipher = Cipher.getInstance("RSA/ECB/OAEPwith" + hashAlgo + "andMGF1Padding", cp);
            cipher.init(Cipher.DECRYPT_MODE, getPrivateKeyFromFile());
            byte[] dec = cipher.doFinal(Base64.getDecoder().decode(base64string));
            return new String(dec);
        } catch (BadPaddingException e) {
            log.error("RsaEncDec | decrypt BadPaddingException", e);
        } catch (NoSuchAlgorithmException | NoSuchPaddingException e) {
            log.error("RsaEncDec | decrypt NoSuchAlgorithmException / NoSuchPaddingException", e);
        } catch (IllegalBlockSizeException e) {
            log.error("RsaEncDec | decrypt IllegalBlockSizeException", e);
        } catch (InvalidKeyException e) {
            log.error("RsaEncDec | decrypt InvalidKeyException", e);
        } catch (InvalidKeySpecException e) {
            log.error("RsaEncDec | decrypt InvalidKeySpecException", e);
        } catch (IOException e) {
            log.error("RsaEncDec | decrypt IOException", e);
        } catch (Exception e) {
            log.error("RsaEncDec | decrypt Exception", e);
        }
        return StringUtils.EMPTY;
    }

    private PublicKey getPublicKeyFromFile() throws IOException, NoSuchAlgorithmException, InvalidKeySpecException {
        byte[] keyBytes = Files.readAllBytes(Paths.get(publicKeyPath));
        X509EncodedKeySpec spec = new X509EncodedKeySpec(keyBytes);
        KeyFactory kf = KeyFactory.getInstance("RSA");
        return kf.generatePublic(spec);

    }

    private PrivateKey getPrivateKeyFromFile() throws IOException, NoSuchAlgorithmException, InvalidKeySpecException {
        byte[] keyBytes = Files.readAllBytes(Paths.get(privateKeyPath));
        PKCS8EncodedKeySpec spec = new PKCS8EncodedKeySpec(keyBytes);
        KeyFactory kf = KeyFactory.getInstance("RSA");
        return kf.generatePrivate(spec);
    }

}

```

Plain Request Json looks like below

```
[
    {
        "custId": 69,
        "mobileNumber": "98765432100",
        "accountNumber": 9876543210,
        "products": [
            {
                "id": "PRAN",
                "type": "FD",
                "status": "Active",
                "deliveryStatus": "pending",
                "issuanceDate": "2025-02-03",
                "expiryDate": "2026-02-03"
            },
            {
                "id": "8989586904",
                "type": "AUTO_LOAD",
                "status": "Active",
                "deliveryStatus": "pending",
                "issuanceDate": "2025-02-03",
                "expiryDate": "2026-02-03"
            },
            {
                "id": "8989586904",
                "type": "SWEEP",
                "status": "Active",
                "deliveryStatus": "pending",
                "issuanceDate": "2025-02-03",
                "expiryDate": "2026-02-03"
            },
            {
                "id": "8989586904",
                "type": "SWEEP",
                "status": "Active",
                "deliveryStatus": "pending",
                "issuanceDate": "2025-02-03",
                "expiryDate": "2026-02-03"
            },
            {
                "id": "8989586904",
                "type": "DIGI_GOLD",
                "status": "Active",
                "deliveryStatus": "pending",
                "issuanceDate": "2025-02-03",
                "expiryDate": "2026-02-03",
                "customField1": "",
                "customField2": "",
                "customField3": "",
                "customField4": "",
                "customField5": ""
            }
        ]
    },
    {
        "custId": 70,
        "natlId": "98765432100",
        "accountNumber": 9876543210,
        "products": [
            {
                "id": "PRAN",
                "type": "FD",
                "status": "Active",
                "deliveryStatus": "pending",
                "issuanceDate": "2025-02-03",
                "expiryDate": "2026-02-03"
            },
            {
                "id": "8989586904",
                "type": "AUTO_LOAD",
                "status": "Active",
                "deliveryStatus": "pending",
                "issuanceDate": "2025-02-03",
                "expiryDate": "2026-02-03"
            },
            {
                "id": "8989586904",
                "type": "SWEEP",
                "status": "Active",
                "deliveryStatus": "pending",
                "issuanceDate": "2025-02-03",
                "expiryDate": "2026-02-03"
            },
            {
                "id": "8989586904",
                "type": "SWEEP",
                "status": "Active",
                "deliveryStatus": "pending",
                "issuanceDate": "2025-02-03",
                "expiryDate": "2026-02-03"
            },
            {
                "id": "8989586904",
                "type": "DIGI_GOLD",
                "status": "Active",
                "deliveryStatus": "pending",
                "issuanceDate": "2025-02-03",
                "expiryDate": "2026-02-03",
                "customField1": "",
                "customField2": "",
                "customField3": "",
                "customField4": "",
                "customField5": ""
            }
        ]
    }
]
```

Request Pojo (To be used in java code)

```java
@Setter
@Getter
@ToString
@Builder
@NoArgsConstructor
@AllArgsConstructor
@JsonInclude(JsonInclude.Include.NON_EMPTY)
@JsonIgnoreProperties(ignoreUnknown = true)
public class Customer {

    @NotNull(message = "is Required")
    @Positive(message = "Must be a number")
    @Max(value=999999999999L, message = "Must not be greater than 15 digits")
    private Long custId;

    @NotEmpty(message = "Natld Cant be null or empty")
    @Pattern(regexp = "^\\d{10,11}$", message = "must be exactly 10 or 11 digits")
    private String mobileNumber;

    @NotNull(message = "is Required")
    @Positive(message = "Must be a number")
    @Min(value = 1000000000L, message = "Must be exactly 10 digits")
    @Max(value = 9999999999L, message = "Must be exactly 10 digits")
    private Long accountNumber;

    @NotEmpty(message = "Products list cannot be empty")
    private List<Map<String, Object>> products;

}
```

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)
	`DIGI_GOLD,`
	`FD,`
	`NCMC,`
	`DEBIT_CARD,`
		50

```
SURAKSHA,
REWARD123,
REWARDMINI,
TRAVELLER,
REWARD123PLUS,
LOAN,
DBT,
FASTAG,
INSURANCE,
APY,
SWEEP,
AUTO_LOAD,
REKYC_DUE_DATE,
UPI_ENABLED,
SMART_PHONE,
SEGMENTATION,
URBAN,
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 ℹ Just in case of extra data needs to be inserted. It can be inserted in custom fields provided	60

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

Api Response:

```
{
    "meta": {
        "code": "1000",
        "description": "Success",
        "status": 0
    },
    "data": [
        {
            "custId": 69,
            "status": 0,
            "message": "Insertion Success"
        },
        {
            "custId": 70,
            "status": 1,
            "message": "Insertion Failed. Will retry"
        },
       {
            "custId": 71,
            "status": 0,
            "message": "Partial Success. Failed to insert  in cache. We Will try to sync in some time"
        }
    ]
}

```

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.

```
{
        "custId": 69,
        "mobileNumber": "98765432100",
        "accountNumber": 9876543210,
        "products": [
            {
                "id": "PRAN",
                "type": "FD",
                "status": "Active",
                "deliveryStatus": "pending",
                "issuanceDate": "2025-02-03",
                "expiryDate": "2026-02-03"
            },
            {
                "id": "8989586904",
                "type": "AUTO_LOAD",
                "status": "Active",
                "deliveryStatus": "pending",
                "issuanceDate": "2025-02-03",
                "expiryDate": "2026-02-03"
            },
            {
                "id": "8989586904",
                "type": "SWEEP",
                "status": "Active",
                "deliveryStatus": "pending",
                "issuanceDate": "2025-02-03",
                "expiryDate": "2026-02-03"
            },
            {
                "id": "8989586904",
                "type": "SWEEP",
                "status": "Active",
                "deliveryStatus": "pending",
                "issuanceDate": "2025-02-03",
                "expiryDate": "2026-02-03"
            },
            {
                "id": "8989586904",
                "type": "DIGI_GOLD",
                "status": "Active",
                "deliveryStatus": "pending",
                "issuanceDate": "2025-02-03",
                "expiryDate": "2026-02-03",
                "customField1": "",
                "customField2": "",
                "customField3": "",
                "customField4": "",
                "customField5": ""
            }
        ]
    }
```