Crypto Guide for External API Clients

The External API Clients has to sign there payload with their private key and encrypt with provided HSBC public key received in secure email, before invoking the API. Similarly only the response will be valid if client application can decrypt the response with there private key and verify the response payload with HSBC public key.

To perform the above operation in Java here is provided the Sample code and instructions. the below provided Java PgpHelper class in java will require the list of dependencies as below.

PgpHelper.java

pom.xml

```
<dependencies>
        <dependency>
            <groupId>org.bouncycastle</groupId>
            <artifactId>bcpkix-jdk15on</artifactId>
            <version>1.56</version>
        </dependency>
        <dependency>
            <groupId>org.bouncycastle</groupId>
            <artifactId>bcpg-jdk15on</artifactId>
            <version>1.55</version>
        </dependency>

        <dependency>
            <groupId>org.apache.commons</groupId>
            <artifactId>commons-lang3</artifactId>
            <version>3.3.2</version>
        </dependency>
        <!-- https://mvnrepository.com/artifact/commons-io/commons-io --
>
        <dependency>
            <groupId>commons-io</groupId>
            <artifactId>commons-io</artifactId>
            <version>2.6</version>
        </dependency>

        <dependency>
            <groupId>org.apache.logging.log4j</groupId>
            <artifactId>log4j-api</artifactId>
            <version>2.11.0</version>
        </dependency>
        <dependency>
            <groupId>org.bouncycastle</groupId>
            <artifactId>bcpg-jdk15on</artifactId>
            <version>1.55</version>
        </dependency>
        <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-databind</artifactId>
            <version>2.9.5</version>
        </dependency>
    </dependencies>
```

The above dependencies will be required from bouncycastle, apache common and logging to make the above class work.

Once this is all set. The client application will be required to configure the following resources.

1. HSBC bank's provided public key on the class path or on absolute path, commonly in resources folder
2. Place the PGP Secret key same as the bank public Key.
3. Configure the password for the client Secret key. as the PGP Secret key is secure place holder for PGP private keys protected by a password.

Once the above is configured. Now you can write the client service class using the following code snippets to complete the following processes.

Load resources

Load the required resources for the process like Own Secret Key and Bank's public key

Load resources

```
BufferedInputStream secret_key= (BufferedInputStream) ClassLoader.
getSystemResourceAsStream("pgp-secret.asc");
BufferedInputStream bankIs = (BufferedInputStream) ClassLoader.
getSystemResourceAsStream("bankpublickey.pgp");
BufferedInputStream dataStream = (BufferedInputStream) ClassLoader.
getSystemResourceAsStream("sampleData.txt");
PGPSecretKey key = pgpHelper.readSecretKey(secret_key);
PGPPrivateKey clientPrivate = pgpHelper.findSecretKey(key, "password".
toCharArray());        // Read the private key from secret
PGPPublicKey clientPublic = key.getPublicKey();
// get client public key from secret too.
PGPPublicKey bankPublicKey = pgpHelper.readPublicKey
(bankIs);                                                        //
read bank key object from the stream
```

Sign and Encrypt the payload

As of now the resources are loaded in application. Now you can use the below code to sign and encrypt the provided payload

Sign and Encrypt

```
PGPSecretKey key = pgpHelper.readSecretKey
(is);
// PGP Helper method to read the client secret from the inputStream.
PGPPrivateKey clientPrivate = pgpHelper.findSecretKey(key, passPhrase.
toCharArray());                        // Get out the client private
key
PGPPublicKey bankPublicKey = pgpHelper.readPublicKey
(bankIs);
// get Bank public Key
ByteArrayOutputStream baout = new ByteArrayOutputStream();
pgpHelper.encryptAndSign(baout,dataStream,bankPublicKey,
clientPrivate);                         // Sign and encrypt the payload
byte[] encAndSigned = baout.toByteArray();
// get the payload out from output stream
baout.close();
// good practice to close the stream
```

At the end of above code the payload is ready in baout an OutPutStream to convert either in string value or write onto the request.

Decrypt and verify the payload

The below code is to decrypt the response from HSBC and verify the signature to ensure the integrity of the content. The below code expect the payload in inputstream shape.

Decrypt and Verify

```
PGPPublicKey bankPublicKey = pgpHelper.readPublicKey
(bankIs);
// PGP Helper method to read bank public key
PGPSecretKey key = pgpHelper.readSecretKey
(is);
// PGP Helper method to read the client secret from the inputStream.
PGPPrivateKey clientPrivate = pgpHelper.findSecretKey(key, passPhrase.
toCharArray());                        // Get out the client private
key

ByteArrayOutputStream bo = new ByteArrayOutputStream();
// place holder to write the decrypted and verified payload
pgpHelper.decryptStream(dataStream, bo, clientPrivate, bankPublicKey,
test);                    // decrypt and verify the payload
bo.close();
```