Annexure A - Common library Specifications

for

Unified Payment Interface

Specifications – Version Draft

DOCUMENT RELEASE NOTICE

Document Details

Name	**Version No.**	**Date**	**Description**
Common Library Specifications for UPI	Draft	28th Sep 2015	Provides Common library specifications for Android

Contents

1. Common Library

1.1 Common library interface

NPCI provides a common library to capture the credentials in a secured way. PSP's will embed the NPCI provided common library in their application. Common library module provides the interface for PSP mobile app in Android, Windows and iOS platforms. Integration methodology and flow will differ depending upon the programming platform used by PSP application.

1.1.1 Interface Requirements

Provided by: Common Library
Used by: PSP Mobile Application
Functionality: PSP mobile app would use the following interface to communicate with common library. It would pass the information required to common library to enable it to capture the sensitive data and return the encrypted data to PSP application.

1.1.2 Input to the common library from the PSP app

Parameter Name	Data Type	Description
keyCode	String	This is a mandatory parameter, specifying whether the NPCI or UIDAI public key is to be used. Valid values can be NPCI or UIDAI.
keyXmlPayload	String	This is a mandatory field containing the digitally signed XML payload received from list-Keys API of UPI. Response of list-keys API not modified
controls	Json	This field specifies the schema for the credential(s) to capture, is in JSON Format. Common Library read this data and generates the Controls. Example is as below:

<table>
<tr><td></td><td></td><td>[
{"name":"otp","label":"Enter your OTP","type":"password","keypad":"normal","validation":"a:4:10"},
{"name":"mpin","label":"Enter your MPIN","type":"text","keypad":"normal","validation":"an:0:6"}
]

This JSON tells the common library that it needs to render the UI for the fields with following details:
<table><tr><th>Cred name</th><th>Label to display</th><th>Display type</th><th>Keypad type</th><th>Data type</th><th>Min. length</th><th>Max. length</th></tr><tr><td>otp</td><td>Enter your OTP</td><td>Password</td><td>Normal</td><td>Alpha</td><td>4</td><td>10</td></tr><tr><td>mpin</td><td>Enter your MPIN</td><td>Text</td><td>normal</td><td>Alpha-numeric</td><td>0</td><td>6</td></tr></table>
If the PSP app passes the label text in different language, it will get displayed in that language.
Based on the number of blocks in the JSON, one or more credential input control will be rendered by the common library.</td></tr>
<tr><td>configuration</td><td>Json</td><td>This is an optional parameter using which the banks/PSPs can customize the UI displayed by the common library. A sample Json can be as below:

{"pspName":"ICICI Bank Ltd.","pspMessage":"Please enter your credentials below.","backgroundColor":"#FF9933","buttonText":"Accept & Continue"}</td></tr>
<tr><td>Salt</td><td>Json</td><td>This is a mandatory parameter that captures different elements of the salt used for encryption. Following is an example of a salt Json:
{ "txnID":"2350406","txnAmount":"29.30"}
Configuration will be maintained in the common library to use one or more of these attributes in the combined string with cred block for different versions of the common library.</td></tr>
</table>

6.1 *Output Response*

Output of the common library will be a HashMap<String, String>(for Android). There can be more than one entries in the HashMap, based on the number of credentials being captured by the common library.

The key of this HashMap will have the credential name. This will use the name of the credential being captured. This should match with the “name” attribute in the control Json in the input.
The value will be a Json string containing the ki(key index) and the encrypted cred block.
Example of the values in the HashMap can be as below:
"mpin" : {"keyId":"9" , "message" : "<Encrypted Message>"}
"otp" : {"keyId":"12" , "message" : "<Encrypted Message>"}

Below code is a sample Android code for integration of thePSP app with the NPCI common library:

```
package org.npci.upi.security.commonsapp;

import android.app.Activity;
import android.app.DownloadManager;
import android.content.Intent;
import android.graphics.Color;
import android.support.v4.app.Fragment;
import android.os.Bundle;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;
import org.npci.upi.security.pinactivitycomponent.PinActivityComponent;

import java.util.HashMap;
import java.util.Map;
import java.util.Random;


/**
 * A placeholder fragment containing a simple view.
 */
public class MainActivityFragment extends Fragment {
    Button button;
    TextView textResult;

    private String encryptedMessage;
    private String keyOrgainzation;
    private String keyID;
    private String xmlPayloadString;


    public MainActivityFragment() {
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup 
container,
                             Bundle savedInstanceState) {

        View rootView = inflater.inflate(R.layout.fragment_main, 
container, false);
        /* Calling app_init() for declaring the event and initialization 
*/
        app_init(rootView);
        return rootView;

    }

    public void app_init(View container){

        button = (Button) container.findViewById(R.id.button);
        textResult = (TextView) container.findViewById(R.id.textView2);
        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(getActivity(), 
PinActivityComponent.class);

                intent.putExtra("keyCode", "ICICI");

                intent.putExtra("xmlPayload", xmlPayloadString);  // It 
will get the data from list keys API response


                try {
                    // This will set the configuration of the app.
                    JSONObject configuration = new JSONObject();
                    configuration.put("pspName", "ICICI Bank Ltd.");
                    configuration.put("pspMessage", "Please enter your 
requested credentials below.");
                    configuration.put("backgroundColor","#FF9933");
                    configuration.put("buttonText", "Accept & 
Continue");
                    //Setting configuration for the app to the intent.
                    Log.i("configuration",configuration.toString());
                    intent.putExtra("configuration", 
configuration.toString());
```

```
                    JSONObject salt = new JSONObject();
                    salt.put("txnId", String.valueOf(new
Random().nextInt(99999999)));
                    salt.put("txnAmount", "29.30");
                    Log.i("salt", salt.toString());
                    intent.putExtra("salt", salt.toString());



                } catch (Exception ex) {
                    ex.printStackTrace();
                }

                JSONArray controlsArray=new JSONArray();
                try {
                    /*** Creating OTP Control Schema ***/
                    JSONObject otpControl = new JSONObject();
                    otpControl.put("name", "otp");
                    otpControl.put("label", "Enter your OTP");
                    otpControl.put("type", "password");
                    otpControl.put("keypad","normal");
                    otpControl.put("validation","a:4:8");
                    controlsArray.put(otpControl);

                    /*** Creating mPin Control Schema ***/
                    JSONObject mpinControl = new JSONObject();
                    mpinControl.put("name", "mpin");
                    mpinControl.put("label", "Enter your MPIN");
                    mpinControl.put("type", "password");
                    mpinControl.put("keypad", "normal");
                    mpinControl.put("validation","an:0:6");
                    controlsArray.put(mpinControl);

                    /*** Creating ccLast4Digit Control Schema ***/

                    JSONObject ccLast4Digit = new JSONObject();
                    ccLast4Digit.put("name", "last4digit");
                    ccLast4Digit.put("label", "Enter your Credit/Debit
Card last 4 digit");
                    ccLast4Digit.put("type", "text");
                    ccLast4Digit.put("keypad","numeric");
                    ccLast4Digit.put("validation","ans:4:4");
                    controlsArray.put(ccLast4Digit);


                } catch (JSONException e) {
                    e.printStackTrace();
                }
                Log.i("Controls JSON", controlsArray.toString());
                intent.putExtra("controls", controlsArray.toString());
                startActivityForResult(intent, 1);
            }
        });
```

```
    }

    @Override
    public void onActivityResult(int requestCode, int resultCode, Intent
data) {
       if (requestCode == 1 && resultCode == Activity.RESULT_OK && data
!= null) {
           HashMap<String, String> credListHashMap = (HashMap<String,
String>)data.getSerializableExtra("credBlocks");
           for(String cred : credListHashMap.keySet()){ // This will
return the list of field name e.g mpin,otp etc...
               try {
                   JSONObject credBlock=new
JSONObject(credListHashMap.get(cred));
                   Log.i("keyId", credBlock.getString("keyId"));
                   Log.i("message", credBlock.getString("message"));

               } catch (JSONException e) {
                   e.printStackTrace();
               }

           }
       }
    }
}
```

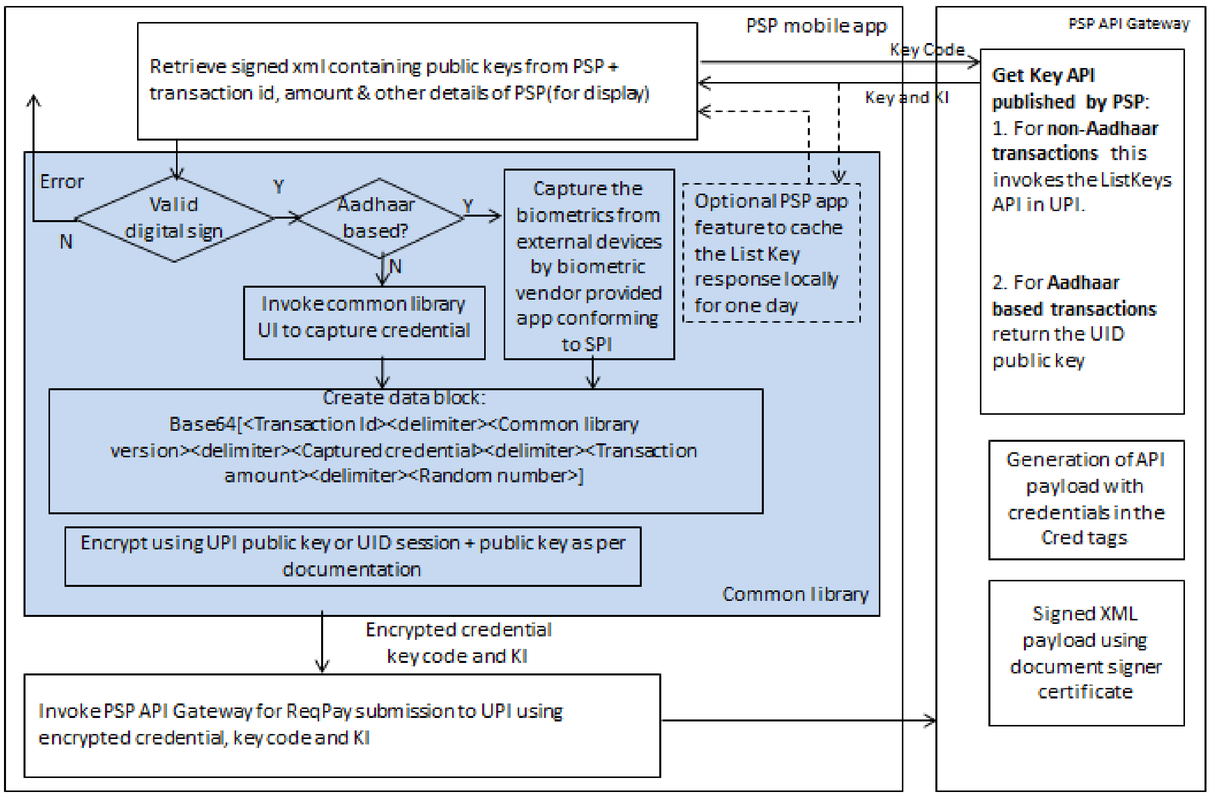
1.2 Encryption Algorithm

The cred block data will be base 64 encoded. The structure of the cred block is mentioned in the next section. RSA 2048 public key encryption will be used to encrypt the base 64 encoded credential block for credentials to be sent to UPI. In case of Aadhaar biometric data capture the data would be encrypted with AES session key and session key would be encrypted with UID public key as per documentation of Aadhaar.

For biometrics capture for aadhaar based authentication, the encryption mechanism is to be consistent with the policy mandated by UIDAI. When it is required to collect fingerprint, the common library will interact with biometric sensor/extractor APIs using their SDKs and obtain minutiae. Then application program forms the PID XML, computes SHA-256 hash of the PID XML, generates a AES-256 random session key, encrypts the PID XML with the session key. Then session key is encrypted using UIDAI's 2048-bit public key. Encrypted PID XML, encrypted session key, and PID Hash (hmac) is sent to UPI server through the PSP server. UPI will route the authentication request to UIDAI.

1.3 Common Library Process Flow

1) PSP mobile application would invoke the common library using the defined interface
2) Common library would first validate the digitally signed xml payload of public keys. If validation is not successful, it will return error accordingly. Otherwise it'll proceed.
3) Common library would pick any one key from the list of keys provided as input. The selection of a single KI will be based on the matching of a random digit among the number of keys available. For example, if five keys are there in the List Keys response, a random number between 1 and 5 will be generated. If the generated number is 4, the key at 4th sequence will be selected for encryption.
    a) If it is Aadhaar based biometric data, common library will choose UID public key, for all other purpose it'll choose UPI public key.
4) Depending upon the input on what credential(s) is(are) to be captured, common library would present a UI to the user. It'll show some basic information about the transaction (passed from PSP).
5) For Aadhaar biometric data capture, common library would invoke the biometric device vendor's mobile SDK conforming to the SPI defined. SPI requirements are described later in this document.
6) The data captured would be prepared in the following way.
    a. Encoded in base64
    b. [<Transaction Id><delimiter><Common library version><delimiter><Captured credential><delimiter><Transaction amount><delimiter><Random number>]
7) Common library would encrypt the base64 encoded data using UPI public key (except Aadhaar biometric data capture). In case of Aadhaar biometric data capture the data would be encrypted with AES session key and session key would be encrypted with UID public key as per documentation of Aadhaar.
8) Common library would return the response to PSP mobile Application.


PSP mobile app
Retrieve signed xml containing public keys from PSP + transaction id, amount & other details of PSP (for display)
Key Code
Key and KI
Error
Valid digital sign
Y
N
Aadhaar based?
Y
N
Invoke common library UI to capture credential
Capture the biometrics from external devices by biometric vendor provided app conforming to SPI
Optional PSP app feature to cache the List Key response locally for one day
Create data block: Base64[<Transaction Id><delimiter><Common library version><delimiter><Captured credential><delimiter><Transaction amount><delimiter><Random number>]
Encrypt using UPI public key or UID session + public key as per documentation
Common library
Encrypted credential key code and KI
Invoke PSP API Gateway for ReqPay submission to UPI using encrypted credential, key code and KI
PSP API Gateway
Get Key API published by PSP: 1. For non-Aadhaar transactions this invokes the ListKeys API in UPI. 2. For Aadhaar based transactions return the UID public key
Generation of API payload with credentials in the Cred tags
Signed XML payload using document signer certificate

1.4 Generation of XML payload in PSP server

Once the PSP server gets the encrypted credentials from the PSP app, it generates the Cred block in the following fashion.

```
<Creds>
    <Cred type="AADHAAR" subtype="IIR|FMR|FIR|OTP">
        <Data> base-64 encoded/encrypted authentication data</Data>
    </Cred>
    <Cred type="OTP" subtype="SMS|EMAIL|HOTP|TOTP">
        <Data> base-64 encoded/encrypted authentication data</Data>
    </Cred>
    <Cred type="PIN" subtype="MPIN">
        <Data> base-64 encoded/encrypted authentication data</Data>
    </Cred>
    <Cred type="CARD" subType="CVV1|CVV2|EMV">
        <Data> base-64 encoded/encrypted authentication data</Data>
    </Cred>
</Creds>
```

1.5 Service Provider Interface(SPI) for Common Library

NPCI will publish a document defining the interface for communication to biometric devices. Any biometric device vendor, intending to be integrated with UPI, must release mobile SDK conforming to the SPI. Common library would invoke the application using the SPI to capture the biometric data from user.

1.6 Versioning Strategy

Common library releases will maintain a version number for different platforms. The UPI server would store all the valid & allowed version numbers in the server. The version number of common library would reside in the encrypted block. At UPI end the same would be validated.

1.7 Configuration Parameters in Common Library

1. Configuration options will be there to maintain the structure (can include reg-ex) / size of the MPIN/password etc. for banks.
2. Configuration is to be maintained on for what credentials what UI component will be displayed. For example, for MPIN entry a text will be displayed. When the

consumer tries to set MPIN, last 6 digits of debit card, OTP and new PIN are to be captured. In this scenario, these three fields are to be displayed by the common library.

3. The structure of the credential block may vary based on common library version. This configuration is to be maintained and validated in UPI.

1.8 Common Library release and distribution

Common library would be distributed as following:

1) Signed jar (AAR) file in case of android
2) Singed dll file in case of windows mobile
3) Signed package in case of iOS

UPI would publish the common library to a specific location to be downloaded by the PSP. A new version of common library release would be needed only in-case of any change in the following: encryption logic / biometric vendor's SPI defined by NPCI / UI to capture data.

1.9 In-app payment/Deep-linking From merchant apps

This is a scenario where Merchant app would use the UPI payment system through a PSP app. This can be done in two ways:

a) A collect call that merchant app would initiate a collect pay request through their server through its PSP (merchant being enrolled with the psp). The end user will respond to the collect call and pay.
b) A consumer uses a merchant app for payment. The merchant app will search for installed apps in the mobile device that can accept payments, using the meta information of these apps. If a PSP app is installed in the device that supports UPI payment channel, payment option through that app will be displayed to the consumer by the merchant app. The merchant app should pass the transaction amount to the PSP app. PSP app will now invoke the NPCI common library to capture the credentials. The end user will review and initiate the payment. The PSP app will communicate with the PSP API gateway, which will subsequently send a ReqPay to UPI along with the cred block. PSP app will communicate the response to merchant app.

In case of android this will be achieved by IMPLICIT Intent. Merchant app would make an Intent request and the PSP app installed in user's mobile would respond to that intent. User will review the submission screen and submit the payment.

1.9.1 Deep-linking configuration guidelines for Android

PSP should configure their app so that merchant app is able to find out their app while doing the payment. The configuration should be done in the following way.

1. **Android Manifest**

```
<activity  android:name=".PayActivity"  android:label="@string/app_name" >
        <intent-filter android:label="@string/pay">
                <action android:name="android.intent.action.VIEW" />
                <category android:name="android.intent.category.DEFAULT" />
                <category android:name="android.intent.category.BROWSABLE" />
                <!-- Accepts URIs that begin with "psp-app://psp-app-pay" -->
                <data  android:scheme="psp-app"    android:host="psp-app-pay"
/>
```

In the above configuration, PayActivity is the class name of the activity which is invoked for the payment. The **data** tag is needed to form the url which the merchant app needs to call to initiate the payment. Each and every PSP app should have the same url so that when the merchant app invokes the payment, the user (payer) would be able to find a list of all available PSP apps in the device.

2. **Pay Activity**

When PSP app receives call from merchant app, it would need to extract different parameters sent by the merchant app and create some transaction request object which would contain the information required to initiate a transaction. It should do it in the Pay Activity.

```
private TransactionRequest request = null;

@Override
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Intent deepLinkingIntent= getIntent();
        // Receive parameters from merchant if called from app
```

```
Intent responseIntent = createResponseIntent(); // Populate response fields which need to
be sent to
                                                //merchant app
activity.setResult(0, responseIntent);
```

Merchant App needs to call the PSP App in the follow way.

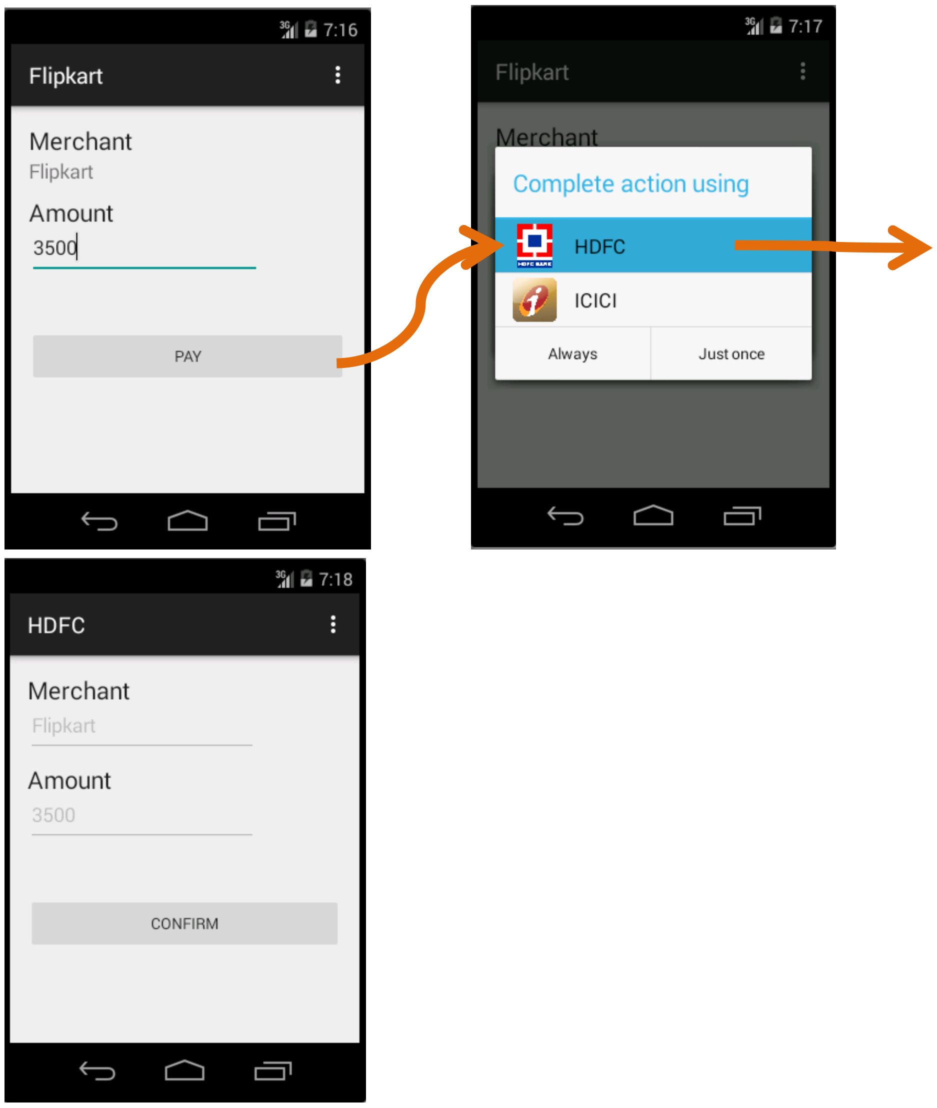
1. **PSP URL**

```
protected static final String PSP_URL = "psp-app://psp-app-pay";
```

2. **On Click Listener of the "Pay using PSP" button**

```
Intent intent = new Intent(Intent.ACTION_VIEW);
populateRequestIntent(intent);
intent.setData(Uri.parse(PSP_URL));
String title = "Pay with";
// Create intent to show chooser. It will display the list of available PSP apps (which have
the same url in
// the maifest)
Intent chooser = Intent.createChooser(intent, title);
// Verify the intent will resolve to at least one activity
```

When user (payer) clicks on the "Pay using PSP" button, the merchant app would create a list of PSPs available in the device which can make the payment. User chooses one of the PSP apps to make the payment. Merchant app invokes the PSP App and expects a response from it.



Merchant app should process the response in the same activity like below.

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent responseIntent) {
        super.onActivityResult(requestCode, resultCode, responseIntent);
        if (requestCode == 1 ) {
                // extract the response code and other fields from the received intent
                processResponseIntent(responseIntent);
```