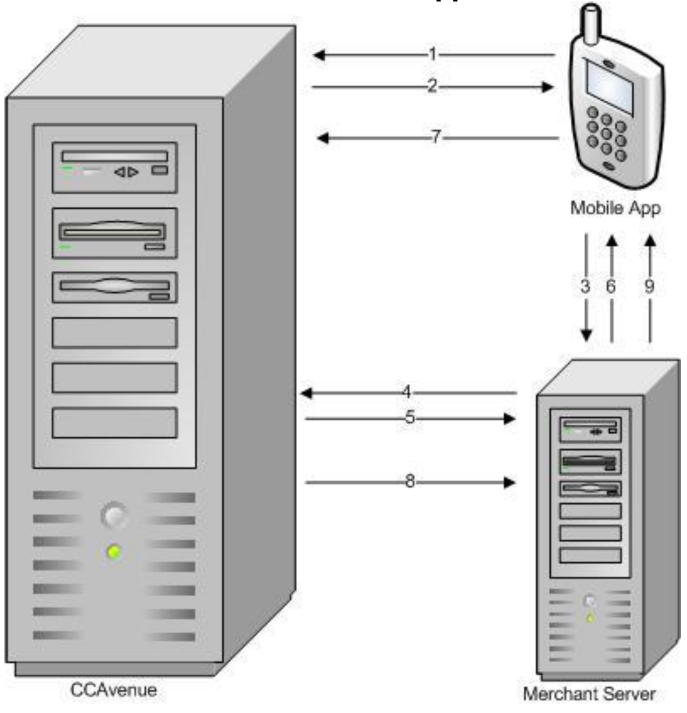
CCAvenue Mobile integration

The CCAvenue mobile integration is designed to enable you to process payments through mobile applications.

Flow of transaction for Mobile App transactions



Steps:

1. Request for payment options from Mobile App to CCAvenue
2. JSON data with payment options returned
3. Request for encryption key from Mobile App to Merchant server
4. Request for encryption key from Merchant server to CCAvenue
5. Dynamic encryption key returned
6. Encryption key sent from Merchant server to Mobile App
7. Transaction with encrypted data sent from Mobile App to CCavenue
8. Processed transaction status returned to Merchant server
9. Transaction status pulled by Mobile App

Integration Types:-

1) Seamless:

If a merchant is configured as seamless then all the billing shipping details including the payment option are accepted on the merchant page. A sample flow for the same is as follows:

a) Merchant will design a screen wherein he will accept all the required fields(billing & shipping details) including the payment options and card details.
b) The required payment options can be fetched by making a json call to the CCAvenue Server by passing required parameters.
c) Once the user enters the required billing and shipping values and the payment option a server to server call to the JSP(GetRSA) will be initiated which is kept on the merchant server for fetching the rsa public key.
d) Using the fetched public key merchant will encrypt parameters(amount,currency,card number, expiry year, expiry month, cvv).
e) After encrypting the parameters merchant will post the billing shipping details and the encrypted value to the CCAvenue server via a browser(embedded webview) post. The user will then be redirected to the selected gateway without displaying CCAvenue's billing shipping screen.
f) Once the user authenticates himself on the bank page the response will be sent by the CCAvenue server to the merchant server on the return url that was configured at the time of registration or the url that was sent in the request.
g) Merchant should then decrypt the response, which can then be parsed to get transaction status.

2) Non-Seamless:

If a merchant is configured as non-seamless then the billing shipping details including the payment option are accepted on the CCAvenue's mobile page. A sample flow for the same is as follows:

a) Merchant will call JSP(GetRSA) which is kept on the merchant server for fetching the rsa public key.
b) Using the fetched public key merchant will encrypt parameters(amount,currency).
c) After encrypting the parameters merchant will post the encrypted value along with the other billing shipping details(if any) to the CCAvenue server via a browser(embedded webview) post.
d) The user will be redirected to the CCAvenue's billing shipping page where he will fill in the required details including the payment option.
e) After filling the required details he will be redirected to the bank page wherein he will authorize himself.
f) After authorization he will be redirected to CCAvenue and a response i.e. Success or failure will be sent to the merchant page which was configured as return url during registration.
g) Merchant should then decrypt the response, which can then be parsed to get transaction status.

Pre-requisites:

1) Access code and encryption key of the registered domain.
2) Merchant must register the public outgoing ip of the server.
3) To use the seamless integration merchant has to submit the PCI certificate or the SAQ based on the no. of hits.
4) This works on Android 4.0 (Ice Cream Sandwich) and above.

URLs:

1) Transaction Url : https://secure.ccavenue.com/transaction/initTrans (To be modified in Mobile App)
2) JSON Url: https://secure.ccavenue.com/transaction/transaction.do (To be modified in Mobile App)
3) RSA Url: https://secure.ccavenue.com/transaction/getRSAKey (To be modified in file GetRSA.jsp)
4) Merchant RSA Url: Path to GetRSA.jsp of merchant Server (To be modified in Mobile App)

Detailed Steps:

To load payment options and cards merchant will initiate a JSON call to server with the below specified parameters:

1) access_code
2) currency
3) amount

The below code does a server to server call for fetching json data. After fetching the required data it is then converted into list objects. The resulting list objects are used to populate values in the spinner object as shown below.

```
// Making a request to url and getting response
List<NameValuePair> params = new ArrayList<NameValuePair>();
// Add required parameters
String jsonStr = sh.makeServiceCall(Constants.JSON_URL, ServiceHandler.POST, params);

//modifying string as per requirement
jsonStr = jsonStr.substring(0,jsonStr.length()-1).replace("processData(","");
JSONArray payOptList = new JSONArray(jsonStr);

// looping through All Payment Options
for (int i=0;i<payOptList.length();i++) {
        JSONObject payOpt = payOptList.getJSONObject(i);
        String payOptStr = payOpt.getString("payOpt");
        try{
                if(payOpt.getString(payOptStr)!=null){
                        payOptionList.add(payOptions.get(payOptStr));//Add payment option only if it includes any
card

                        JSONArray cardArr = new JSONArray(payOpt.getString(payOptStr));
                        for(int j=0;j<cardArr.length();j++){
                                JSONObject card = cardArr.getJSONObject(j);
                                CardTypeDTO cardTypeDTO = new CardTypeDTO();
                                // set values in the above dto and add the dto to respective list objects
```

```
                    }
            }
        }catch (Exception e) {}
}
```

Note: The parameters such as accessCode, amount, currency are fetched from previous activity in the above code.

After populating the list, the values can be populated in the spinner object using the following code

```
// bind adapter to spinner
Spinner payOpt = (Spinner) findViewById(R.id.payopt);
PayOptAdapter payOptAdapter = new PayOptAdapter(m,android.R.layout.simple_spinner_item, payOptionList);
payOpt.setAdapter(payOptAdapter);
```

Now when a user selects an item from the payment option dropdown a corresponding list is to be populated in the card type dropdown. The following listener is to be added for the same.

```
//set a listener for selected items in the spinner
payOpt.setOnItemSelectedListener(new OnItemSelectedListener(){
        @Override
        public void onItemSelected(AdapterView parent, View view, int position, long        id) {
                //Load respective card in the cardType spinner and add an item listener for cardType.
        }
        @Override
        public void onNothingSelected(AdapterView<?> parent) {}
});
```

Once the user enters the required billing shipping details and selects a payment option and the related card type, user will click on the pay button. The onclick event of the pay button will pass the values to the next activity in the following way.

```
public void onClick(View view) {
        //Do validations and sent the required parameters to the following activity.
}
```

The first thing to do in the web view activity is to fetch RSA public key by requesting the page that is kept on merchant server. To fetch RSA public key in app, parameters(accessCode and orderId) are to be sent to the JSP(GetRSA). The GetRSA JSP will then intitiate a server to server call to the CCA server with those parameters and will get an RSA public key in response which will be forwarded to the app. Using the RSA key the following parameters are to be encrypted:

1) Card Number(Only in case of credit/debit cards)
2) Card Expiry Month(Only in case of credit/debit cards)
3) Card Expiry Year(Only in case of credit/debit cards)
4) Card CVV(Only in case of credit/debit cards)
5) Amount
6) Currency

```
// Creating service handler class instance
ServiceHandler sh = new ServiceHandler();

// Making a request to url and getting response
List<NameValuePair> params = new ArrayList<NameValuePair>();

// Add required params

String vResponse =
sh.makeServiceCall(mainIntent.getStringExtra(AvenuesParams.RSA_KEY_URL),ServiceHandler.POST, params);
if(!ServiceUtility.chkNull(vResponse).equals("")
                                && ServiceUtility.chkNull(vResponse).toString().indexOf("ERROR")==-1){
        // form a string including all the above specified parameters as given in the integration kit
        encVal = RSAUtility.encrypt(vEncVal.substring(0,vEncVal.length()-1), vResponse);
}
```

CCAvenue®

Once the details are encrypted those are sent in the request while opening up a web view.

```
webview.postUrl(Constants.TRANS_URL, EncodingUtils.getBytes(vPostParams, "UTF-8"));
```

After forming the parameter string it is to be posted to the url using the postUrl method of the webview as shown above.

Once the webview is successfully opened, the user will be redirected to the bank page. After completing the authorization process the bank will send the response to the CCAvenue Server. The response will then be parsed to determine the status and same will be posted to the return url. To read the response merchant need to decrypt the parameters using the provided AesCryptUtil class. To monitor the flow and read the html source of the page a listener must be added in the app.

```
@SuppressWarnings("unused")
class MyJavaScriptInterface{
        @JavascriptInterface
        public void processHTML(String html){
                // process the html as needed by the app
        }
}
```

The above listener is to be registered with the webview which will read the html page to understand the status of the transaction. Listener is to be registered as specified below:

```
final WebView webview = (WebView) findViewById(R.id.webview);
webview.getSettings().setJavaScriptEnabled(true);
webview.addJavascriptInterface(new MyJavaScriptInterface(), "HTMLOUT");
webview.setWebViewClient(new WebViewClient(){
        @Override
        public void onPageFinished(WebView view, String url) {
                super.onPageFinished(webview, url);
                if(url.indexOf("/ccavResponseHandler.jsp")!=-1){
webview.loadUrl("javascript:window.HTMLOUT.processHTML('<head>'+document.getElementsByTagName('html')[0].innerHTML+'</head>');");
            }
        }

        @Override
        public void onReceivedError(WebView view, int errorCode, String description, String failingUrl) {
                // raise error
        }
});
```

The above specified listener is to be invoked only on the merchant page. In order to achieve that an anonymous class WebViewClient is to be set using the setWebViewClient method of the webview. The method onPageFinished should be overridden to specify our implementation which will check for the url and will then invoke the javascript method to get the html of that page as shown above.

After determining the status of the transaction an acknowledgement page can be displayed to the user by sending these parameters to other activity as shown below.

```
Intent intent = new Intent(getApplicationContext(),StatusActivity.class);
                                        intent.putExtra("transStatus", status);
                                        startActivity(intent);
```