Integrate 3D Secure 2.0 (Direct or CSE)

INDEX

Introduction to 3D Secure

3D secure is an additional layer of verification for Card not present (CNP) transactions.
There are 2 versions available for 3D secure:

> **3D Secure 1.0:** In 3D secure 1.0, customers are redirected to the issuer's website to provide additional authentication data such as password or SMS verification code.
>
> **3D Secure 2.0:** In 3D secure 2.0, customers are not redirected to issuer's website but performs authentication within a merchant's website or mobile app using passive, biometric, and two-factor authentication approaches.3-D Secure 2.0 is compliant with EU "strong customer authentication (SCA)" mandates.

Combined with confusing web redirect experiences which made 3DS 1.0 fail customers and businesses. Not only did 3DS 1.0 lack native in-app and web flows, but it also introduced confusing and difficult-to-remember authentication prompts.This resulted in legitimate customers dropping out of the payment flow.

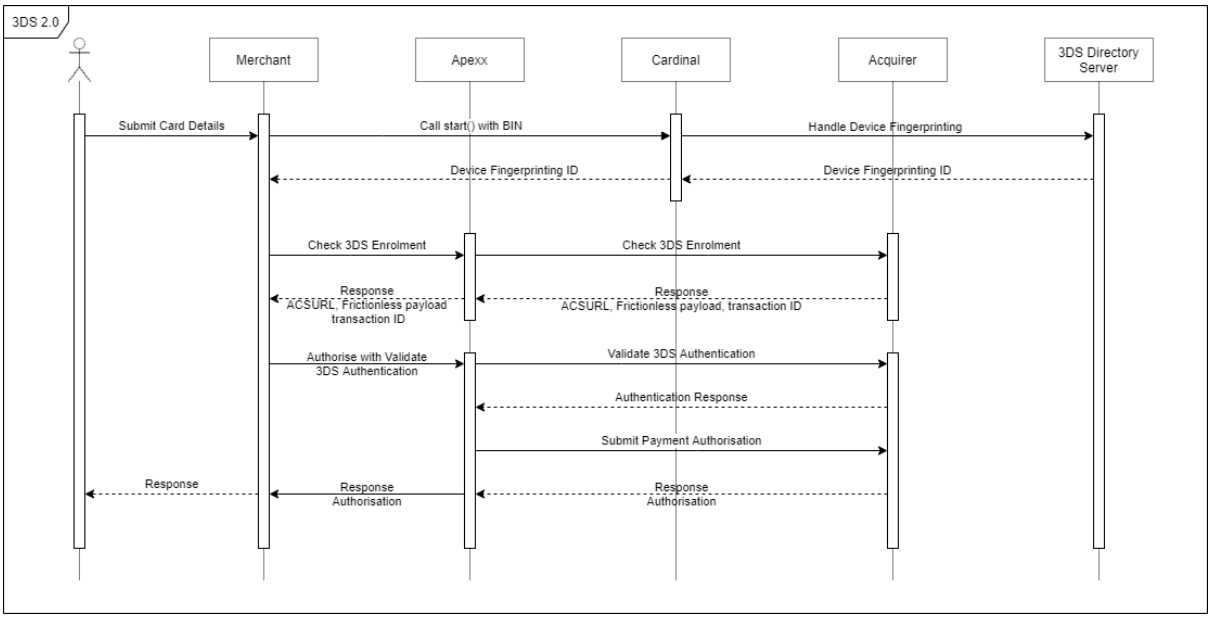
3D Secure 2.0 addresses many of 1.0's issues, while bringing benefits across a wider set of use cases for businesses all over the world. By supporting additional data during transactions, risk-based decisions will be possible on whether to authenticate or not.

3D Secure 2 Implementation Flow

3DS 2.0 transactions can go through either a frictionless flow or challenge flow based on the requirements of the issuers.

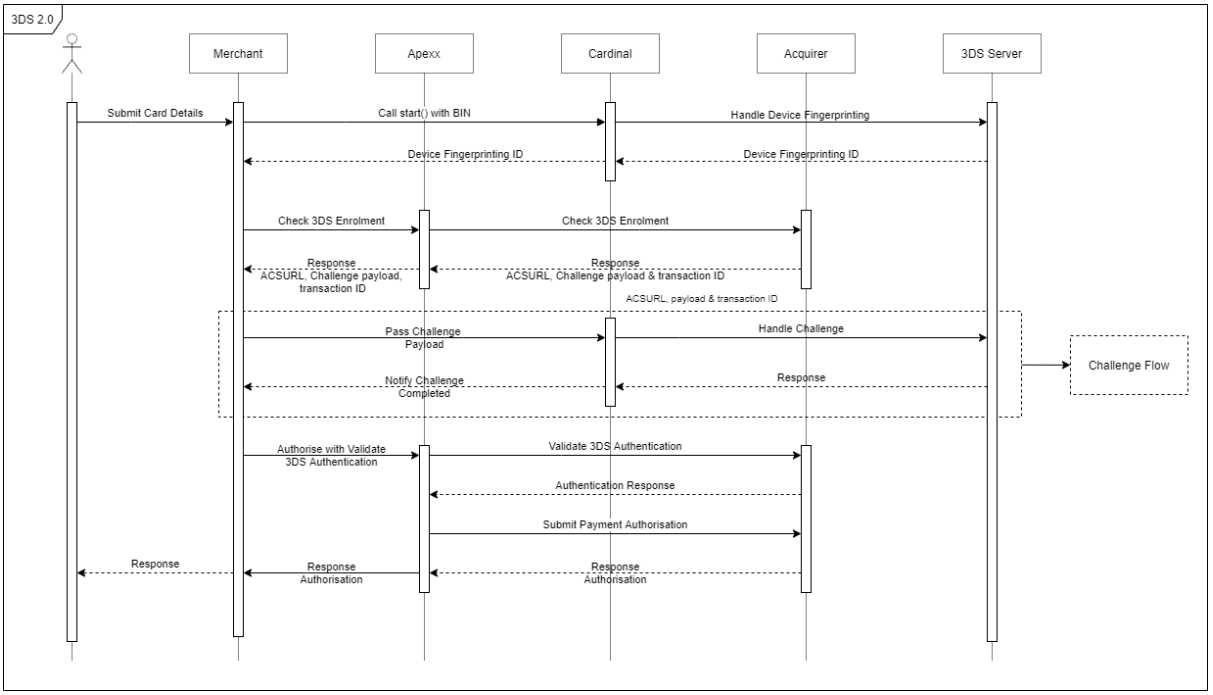
Frictionless Flow

Frictionless flow allows a transaction to be processed without additional intervention from the card holder for the authentication process by exchanging all necessary information in the background using the shopper's device fingerprint.



Challenge Flow

Challenge flow requires a customer's intervention for the authentication process based on secure customer authentication factors.



CSE - Client Side Encryption

- Client Side Encryption (CSE) is a card payment integration method that gives merchants full flexibility with how the present card collection fields to their customers whilst allowing merchants to minimise their exposure when it comes to PCI Compliance.
- CSE integration merchants to be PCI DSS compliant to a level of Self Assessment Questionnaire A-EP or above, in accordance with the latest PCI DSS standards (v3.2.1).
- Your particular level of compliance may differ depending on your own unique setup, so we recommend consulting a QSA to make sure you're set up in the right way to meet your desired PCI DSS level.

Integration

APEXX supports integration for 3DS 2.0 via Cardinal Cruise Hybrid Integration method.

Prerequisite

Cardinal Cruise Account Credentials

As part of registering for Cardinal Cruise, we receive 3 values that are used to authenticate yourself with Cardinal. JWT creation requires that these values have been generated and handed off to the Cardinal. We will provide you with these details during the onboarding process based on your preference.

1. **API Identifier** - A non-secure value that should be passed within the JWT under the iss claim.
2. **Org Unit Id** - a non-secure value that should be passed within the JWT under the OrgUnitId claim.
3. **API Key** - A secure value that should only ever be known between you and Cardinal. This value should never be rendered or displayed anywhere your users could find it. The API Key should only be used to sign the JWT and to verify a JWT signature from Cardinal. It should never be included within the JWT itself.

Apexx Account Credentials

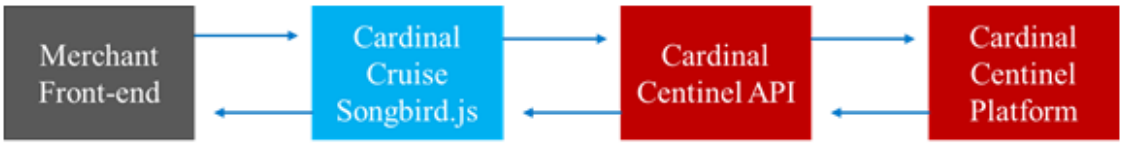
Apexx account credentials like account_id and X-APIKEY provided to you by the implementation manager. Please ask your implementation manager to enable your accounts for processing 3d secure 2.0.

Client Side Encryption JS and keys

Contact your implementation manager to provide you the Encryption key which will be used for encrypting the payment details. CSE javascript can be downloaded from here.

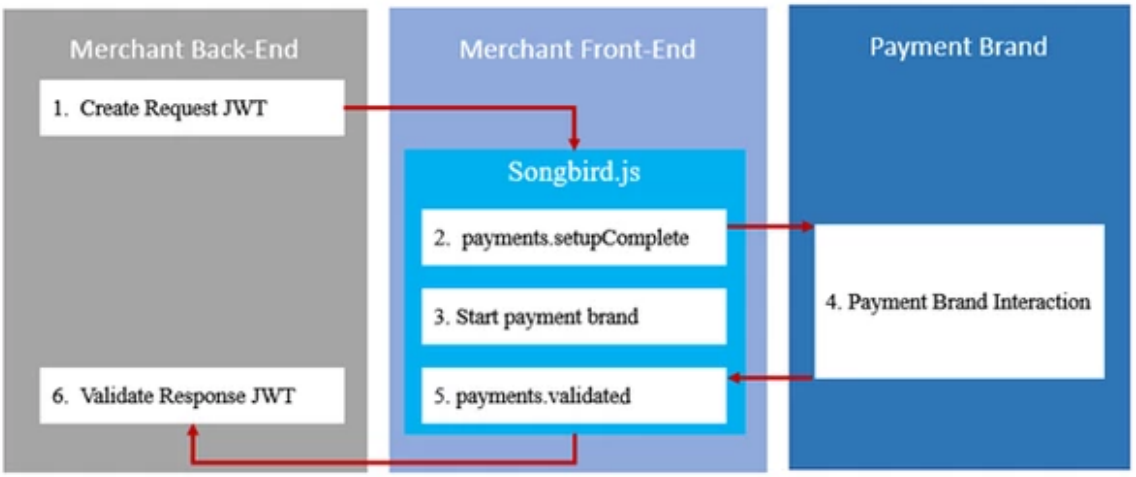
How It Works

Cardinal Cruise uses Songbird.js to perform most of the heavy lifting on behalf of our customers. It will handle all device data collection and communication with our Cardinal Centinel platform, as well as the user experience to the consumer for both CCA and Alternative Payment brands.



- Because Songbird is a client side JavaScript library it can only interact with payment brands client side. Any interactions that require server side implementations - such as our Cardinal Cruise Hybrid integration - are out of scope for Songbird.js, and the merchant may need to integrate to the Cardinal Centinel platform directly.

- The diagram below shows the high-level event execution order of a transaction from the merchant's perspective. The enumerated events do not map to the integration steps found within this document. The enumerated events are simply to help illustrate the order events occur during a transaction.



Integration Overview

1. JWT(JSON Web Token) Creation
2. Include the Script

3. Configure It
4. Listen for Events
5. Initialize It
6. Include BIN Detection
   Points 7 to 10 are applicable for CSE integration only:
7. *Encryption key*
8. *Include the Apexx CSE javascript Library*
9. *Set the public key (Encryption Key)*
10. *Implement the form*
11. Lookup Request (Enrollment)
12. Handling the Lookup Response (Enrollment Response)
13. Continue with CCA(Cardinal's Consumer Authentication)
14. Handling the CCA Response
15. Validate JWT
16. Verify Authentication
17. Authorisation

3DS 2.0 Integration Steps

1. Create JWT(JSON Web Token)

- **JSON Web Token (JWT)** is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed. JWTs can be signed using a secret (with the HMAC algorithm) or a public/private key pair using RSA or ECDSA.
- Cardinal Cruise utilizes a JWT to handle authentication and to assist in passing secure data between you and Cardinal.
- The JWT is a JWS with the signature using a SHA-256 HMAC hash algorithm.
- The JWT must be created server-side and sent to the front end to be injected into the JavaScript initialization code.
- Creating a JWT client-side is not a valid activation option. Each order should have a uniquely generated JWT associated with it.
- You can create JWT with your preferable server side programming language like php,java,.net. Etc
- **Security Warning :** For security reasons, all JWT creation must be done on the server side. When creating the JWT, use your company API Key as the JWT secret.

JWT Fields:

1. **Mandatory Fields:** Below fields are mandatory for creating JWT.

Name	Description
jti	JWT Id - A unique identifier for this JWT. This field should change each time a JWT is generated.
iat	Issued At - The epoch time in seconds of when the JWT was generated. This allows us to determine how long a JWT has been around and whether we consider it expired or not.
iss	Issuer - An identifier of who is issuing the JWT. We use this value to contain the Api Key identifier or name.
OrgUnitId	The merchant SSO OrgUnitId
Payload	The JSON data object being sent to Cardinal. This object is usually an Order object

2. **Optional Fields:** Below fields are optional for creating JWT.

Name	Description
ReferenceId	This is a merchant supplied identifier that can be used to match up data collected from Cardinal Cruise and Centinel. Centinel can then use data collected to enable rules or enhance the authentication request.
ObjectifyPayload	A boolean flag that indicates how Centinel Api should consume the Payload claim. When set to true, this tells Centinel Api the Payload claim is an object. When set to false, the Payload claim is a stringified object. Some Jwt libraries do not support passing objects as claims, this allows those who only allow strings to use their libraries without customization
exp	Expiration - The numeric epoch time that the JWT should be consider expired. This value is ignored if its larger than 2 hrs. By default we will not consider any JWT older than 2 hrs.

JWT creation example: Link: JWT_ Creation

The below code samples are to provide you with a basic idea on how to generate JWT.

Example payload :

```
{
  "jti": "19ec6910-5048-11e6-8c35-8789b865ff15",  //UUID
```

```
 "iat": 1577091197, //JWT initialisation time
 "iss": "<API Identifier >",
 "OrgUnitId": "<Org Unit Id>",
 "Payload": {
  "OrderDetails": {
   "OrderNumber": "19ec6910-5048-11e6-8c35-8789b865ff15" //your
transactionId
  }
 },
 "ObjectifyPayload": true,
 "exp": 1577096197 //JWT expiration time
}
```

Please find the field details for the Payload object from the below link: **https://cardinaldocs.atlassian.net/wiki/spaces/CC/pages/32950/Request+Objects**

JWT library:
You can find JWT libraries as per your requirement from here **https://jwt.io/#libraries-io**.

Note: A JWT is composed of 3 sections of URL base64 encoded strings of data separated by a period. The first 2 sections are JSON objects while the last section is a digital signature that has been URL base64 encoded.

JWT Sample:

```
eyJhbGciOiJIUzI1NiJ9.eyJqdGkiOiIxOWVjNjkxMC01MDQ4LTExZTYtOGMzNS04
Nzg5Yjg2NWZmMTUiLCJpYXQiOjE1NzcwOTExOTcsImlzcyI6IjVkYmFiYTNjYWZ
hODBkMWQ5OGY3OGZmOCIsIk9yZ1VuaXRJZCI6IjVkMjJmODMzZTA5MTlmM
DkyNDdhZDhmMyIsIlBheWxvYWQiOnsiT3JkZXJEZXRhaWxzIjp7Ik9yZGVyTnVt
YmVyIjoiMTllYzY5MTAtNTA0OC0xMWU2LThjMzUtODc4OWI4NjVmZjE1In19LCJ
PYmplY3RpZnlQYXlsb2FkIjp0cnVlLCJleHAiOjE1NzcwOTYxOTd9.TsAvQJAYkQx
de4yrogiwzIcpNDHh2GXV68b0p3qCgvY
```

Header : A small JSON object that specifies what type of algorithm was used to generate the digital signature. This section is denoted in the above example in **red**.

Payload : A JSON object that contains the data being sent from one party to another. This is where all request data the merchant sends to Cardinal lives and vise Versa. This section is denoted in the above example in **green**.

Signature : A URL base64 encoded the hash value of the header and payload. This is used to verify the contents of the JWT have not been tampered with. Signatures are generated using a shared secret value so

only the creator and consumer have the ability to create the signature. As long as the shared secret stays a secret between the 2 parties no one should be able to spoof the signature. Signatures are verified by the consumer recreating the hash value and checking that it matches the one attached to the JWT. This section is denoted above in **blue**.

Decrypt JWT: You can use this link **https://jwt.io/**.

Conclusion: So based on the above information you can create JWT from payload data.

2. Include the Script in Hosted payment page

- In this step, You need to import songbird.js in your Hosted payment page.
- Add Songbird.js to your site just like any other client-side script - through a script tag.

Test environment :

```
<script
src="https://songbirdstag.cardinalcommerce.com/cardinalcruise/v1/songbird.js"></script>
```

Production environment :

```
<script
src="https://songbird.cardinalcommerce.com/cardinalcruise/v1/songbird.js"></script>
```

3. Configure It

- In this step, You have to configure **Cardinal.configure()** in your JavaScript file.
- Using the **Cardinal.configure()** function allows you to pass a configuration object into Songbird.
- There are different configuration options that can be used to initialize Songbird through Cardinal.configure function. You can call this function once per page load. you can configure your root level configuration like logging, button, payment, etc.

Example:

```
Cardinal.configure({
        logging: {
```

```
                level: "on"  // on - Enable logging
            }
    });
```

4. Listen for Events

- In this step, You have to trigger below two events.
- This function sets up an event subscription with Songbird to trigger a callback function when the event is triggered by Songbird.
- You can subscribe to an event using the **Cardinal.on()** function structured like:
  Cardinal.on(EVENT_NAME_SPACE,CALLBACK_FUNCTION);

Example:

```
Cardinal.on('payments.setupComplete', function(){
        // Do something
});
```

payments.setupComplete

- This optional event triggers when Songbird has successfully initialized, after calling the **Cardinal.setup()** function.
- This event will not trigger if an error occurred during your Cardinal.setup() call (this includes a failed JWT authentication) and this event will only trigger once per page load. If your callback gets executed, you know that Songbird is available to run transactions.
- This function will receive 2 arguments that describe the loaded state of Songbird and the current session identifier.
- You will get sessionId from this event. It will be used in further steps.

Example:

```
Cardinal.on('payments.setupComplete',
function(setupCompleteData) {
        var  sessionId = setupCompleteData.sessionId;
});
```

payments.validated

- This event is triggered when the transaction has been finished from the Cardinal end. This is how Songbird hands back control to your webpage.
- This event is triggered with the alternative payment brand results that require a payment authorization. The ActionCode (SUCCESS, NOACTION, FAILURE, ERROR) field should be used to determine the overall state of the transaction.
- Additional information can be found in the fields ErrorNumber and ErrorDescription if the ActionCode indicates an issue was encountered.

Example:

```
Cardinal.on("payments.validated", function (data, jwt) {

    switch(data.ActionCode) {
        case "SUCCESS":
        // Handle successful transaction, send JWT to backend to verify(JWT verification is optional), Then need to call Apexx 3DSVerifyAuthentication API and handle the response of it as a payment response. It is a final authorisation call to the gateway.
        break;
        case "NOACTION":
        // Handle no actionable outcome
        break;
        case "FAILURE":
        // Handle failed transaction attempt
        break;
        case "ERROR":
        // Handle service level error
        break;
    }
});
```

5. Initialize It

- In this step, You need to pass a jwt token(that you have created in the 1st step).
- **Cardinal.setup()** will initiate the communication process with Cardinal.
- You should only call this function once per page load.
- A common way to pass your JWT into this JavaScript function is to place the JWT value into a hidden input on page load. Within Cardinal.setup(), you can then look for that element and select its value.

JWT hidden field in the HTML :

```
<input type="hidden" id="JWTContainer" value="[Insert your JWT here]" />
```

Cardinal Setup :

```
Cardinal.setup("init", {
    jwt: document.getElementById("JWTContainer").value
});
```

- **Note:** Cardinal.setup does not catch errors. Use the **payments.validated event** mentioned in **Step 4** instead.

6. Include BIN Detection

- In this step, You need to trigger a **bin.process** event with a card number.
- The merchant will need to provide a minimum of the first 6 (e.g. BIN) up to the full card number of the consumer (e.g. max of 19 digits).
- This event will wait for a static amount of time before resolving to allow for device data collection to complete. The implementor should wait for this event to complete before moving forward with a transaction.
- This event will need to occur before the **Lookup Request (Enrollment).**
- You can check the bin details whether it is correct or not and a way to specifically run the EMV 3DS Method URL.
- If no Method URL was provided, Cardinal will not perform additional device data collection at this time.

- The **bin.process** event will return a promise that will be fulfilled when the EMV 3DS Method URL 'attempt' has been completed. The promised result will include a single object containing the result of the 3DS Method URL attempt.

Example:

```
Cardinal.trigger("bin.process", 520000)
.then(function(results) {

  if(results.Status) {
     // Bin profiling was successful.
       console.log("Bin profiling successful...");
//You can put enrollment call here

  } else {
     // Bin profiling failed
     console.log("Bin profiling failed...");
  }
 }).catch(function(error){
  // An error occurred during profiling
    console.log("Exception in Bin profiling ");
 });
```

For more details **bin.process** : https://cardinaldocs.atlassian.net/wiki/spaces/CC/pages/557065/Songbird.js#Songbird.js-bin.process

- **You can verify the result.Status by using console.log("verify message"). It's optional.**

Below Steps from 7 to 10 are required for Client-Side Encryption Implementation only

7. Encryption key

- The encryption key must be supplied to the Client-Side Encryption library and is used to encrypt the sensitive card details.
- The decryption key exists within a secure environment at APEXX.
- Contact Apexx Implementation manager for the encryption key.

8. Include the APEXX CSE JavaScript Library

- For CSE to work, reference the APEXX CSE JavaScript library. The way to do this. Automatically reference the latest version of the current major release.

Test environment :

```
<script
src="https://sandmgw.apexxfintech.com/mgw/resources/js/cse/apexx-cse-1.0.js"></script>
```

Production environment :

```
<script
src="https://production.apexxfintech.com/mgw/resources/js/cse/apexx-cse-1.0.js"></script>
```

- Download our Javascript SDK from https://sandmgw.apexxfintech.com/mgw/resources/apexx-cse-1.0.zip

9. Set the Public Key (Encryption Key)

- Set your Public Key (to identify you on our server) in your Javascript by below function:

```
APEXX.setPublicKey("YOUR PUBLIC KEY");
```

10. Implement the form

- For the form to work, use the "data-apexx" attribute for each item of cardholder data.

Example:

```
<body>
    <form action="[YOUR FORM ACTION]" method="post"
data-apexx="payment-form">
      <input data-apexx="card_holder_name"> <!-- This is the cardholder
name field -->
      <input data-apexx="card_number"> <!-- This is the card number
field -->
```

```
    <input data-apexx="exp_month"> <!-- This is the expiry month field
-->
    <input data-apexx="exp_year"> <!-- This is the expiry year field -->
    <input data-apexx="cvv"> <!-- This is the CVV field-->
    <input type="hidden" name="[YOUR DATA PARAMETER]"
data-apexx="encrypted_data">  <!-- This is the hidden field for
encrypted_data -->
    <input type="hidden" name="maskedCardNumber"
data-apexx="masked_card_number">  <!-- This is the hidden field for
masked_card_number -->
    <input type="submit">
  </form>
</body>
```

- In this example the Apexx CSE library will set the encrypted data in data-apexx="encrypted_data" element.

How it all fits together

- Include the APEXX CSE Javascript file & Set the Public Key in the head of the HTML page.

```
<head>
  <script type="text/javascript"
src="https://sandmgw.apexxfintech.com/mgw/resources/js/cse/apexx-
cse-1.0.js"></script>
  <script type="text/javascript">
   window.onload = function() {
    APEXX.setPublicKey("YOUR PUBLIC KEY");
   }
  </script>
</head>
```

- Include the apexx payment form html in the card details collection elements of your payment page.

```
<form action="[YOUR FORM ACTION]" method="post"
data-apexx="payment-form">
  <input data-apexx="card_holder_name">
  <input data-apexx="card_number">
  <input data-apexx="exp_month">
  <input data-apexx="exp_year">
  <input data-apexx="cvv">
```

```
   <input type="hidden" name="[YOUR DATA PARAMETER]"
data-apexx="encrypted_data">
   <input type="hidden" name="maskedCardNumber"
data-apexx="masked_card_number">
   <input type="submit">
 </form>
```

Below steps will be the flow to process a CSE payment with APEXX:

a. Include APEXX.js(apexx-cse-1.0.js) into your page.
b. Implement the HTML form as mentioned above.
c. On pressing of Submit/Pay button, our JS(apexx-cse-1.0.js) will take the data i.e card_number, exp_month, exp_year, cvv, etc. and do the encryption and JS file will set the encrypted data into hidden parameter and call to form action (merchant server-side)

3. Call APEXX /payment/direct API with encrypted value along with other information. (Server(Merchant) to server(APEXX) API call).i.e:

```
{
   "account":"your _account_id",
   "organisation":"",
   "currency":"",
   "amount":1500,
   "capture_now": false,
   "card": {
      "encrypted_data" : "{encrypted_value}"
   },
   "dynamic_descriptor": "Your Order",
   "customer": {
      "customer_id" : "",
      "last_name" : "",
      "postal_code" : "",
      "account_number" : "",
      "date_of_birth" : "1993-07-01"
   },
   "customer_ip": "192.168.1.141",
   "billing_address": {
      "first_name": "testfirstname",
      "last_name": "testlastname",
      "email": "name@apexx.global",
      "address": "5a Underwood Street",
      "city": "London",
      "state": "London",
      "postal_code": "N17LY",
```

```
      "country": "GB",
      "phone": "07437836162"
   },
   "merchant_reference": "your_reference",
   "recurring_type": "first",
   "user_agent": "Mozilla/5.0 (Windows; U; Windows NT 6.1; en-GB;rv:1.9.2.13)
Gecko/20101203 Firefox/3.6.13 (.NET CLR 3.5.30729)",
   "webhook_transaction_update": "https://yourwebhook.com/",
   "shopper_interaction": "ecommerce",
   "three_ds": {
      "three_ds_required" : "true",
            "three_ds_version": "2.0",   // Possible values: 1.0, 2.0
            "reference_id": "Cardinal Session Id"
   },
   "card_brand":"visa"
}
```

CSE Fields with validation function

Filed Name	Possible values	Description	Validation function

Card Holder Name	Only alphabetic characters, Special characters allowed are (hyphen(-), underscore(_), dot(.), comma(,) and apostrophe('))	It is representing the card holder name and it's optional field.	isValidCardHolderName([CARD HOLDER NAME]);
			Success: {"isError":false}
			Failed: {
			"errorMessage":"Please enter valid card holder name",
			"isError":true
			}

Card Number	12-19 Digit	Card number used for the transaction, also known as PAN.	**Characters Validation :**
			isValidCardNumber([CARD NUMBER]);
			
			Card Number Validation with Luhn check:
			luhnCheck([CARD NUMBER]);
			Success: {"isError":false}
			Failed: {
			"errorMessage":"Please enter valid card number",
			"isError":true
			}

Expiry Month	2 Digit (01-12)	Card expiry month. A string representing the month, valid values are 01 to 12.	isValidMonth([EXPIRY MONTH]);
			
			Success: {"isError":false}
			Failed: {
			"errorMessage":"Please enter valid Expiry Month",
			"isError":true
			}

Expiry Year	2 Digit	Card expiry year. A string representing the last two digits of the year, e.g. 19 for 2019.	isValidYear([EXPIRY YEAR]);
			
			Success: {"isError":false}
			Failed: {
			"errorMessage":"Please enter valid Expiry Year",
			"isError":true
			}

CVV	3-4 Digit	The Card Verification Value. Note: CVV is not required when a transaction is a recurring or one click transaction.	isValidCVV([EXPIRY CVV]);
			
			Success: {"isError":false}
			Failed: {
			"errorMessage":"Please enter valid CVV",
			"isError":true
			}

.

- **Conclusion:** So based on the above information you can integrate with CSE.
- Now you have CSE encrypted data, You need to pass it for Lookup Request(Apexx Enrollment).

11. Lookup Request (Enrollment)

- In this step, You need to make a first request of the transaction that is used to process the Consumer Authentication transaction request.
- You must ensure that Cardinal has the proper Acquiring Merchant Identification Number and corresponding Acquiring Bank Identification Numbers (BINs) configured in your account.
- For the lookup request, you need to call our **3DSEnrol API** https://sandmgw.apexxfintech.com/mgw/v2/api/doc#operation/cardPayWithThreeDSUsingPOST with some extra fields like below:
  - You need to pass your **encrypted data** in the **card block** for the Enrollment process.
  - Here **reference_id** will be the **sessionId** that we received in the "**payments.setupComplete**" event.

Example:

```
{
.....
// Add 3DSEnrol API request
.....
"card":{
"encrypted_data": "string"
}
  .....
   // 3DS 2.0 fields in three_ds block
```

```
.....
"three_ds":{
  "three_ds_required" : "true",
  "three_ds_version": "2.0",     // Possible values: 1.0, 2.0
  "reference_id": "string" // it should be sessionId of cardinal
  }
}
```

12. Handling the Lookup Response (3DSEnrol API Response)

- The Lookup Response will return an acsURL, paReq, reference_id, authentication_transaction_id, three_ds_enrollment, three_ds_version and specification_version field.
- The lookup response verifies that the transaction is eligible for Authentication or not by the three_ds_enrollment field.
- If the **three_ds_enrollment** field contains a **true** value, then Authentication is available and you need to redirect to the **ACSUrl** to complete the transaction.
- If the **three_ds_enrollment** field value is **false**, then you are NOT eligible for Authentication.
- After calling **Apexx enrollment API**(**3DSEnrol API**), you will receive a lookup(Enrollment) response like below:

Example:

```
{
        "_id": "string"
      "created_at": "string",
      "three_ds":
      {
            "three_ds_required": true,
            "three_ds_version": "2.0",
            "acsURL": "string",
            "paReq": "string",
            "three_ds_enrollment" : true,
            "reference_id": "string",
            "authentication_transaction_id": "string",
            "directory_server_transaction_id":"string",
            "specification_version": "string"
       },

}
```

13. Continue with CCA

- After the Lookup Response is returned, take the acsURL, Payload, and TransactionId and include them in the **Cardinal.continue** function in order to proceed with the authentication session.
- The Cardinal.continue will display a modal window and automatically post the consumer's session over to the acs url for authentication.

Note:

Request fields	Enrollment response fields
AcsUrl	acsURL
Payload	paReq
transactionId	authentication_transaction_id

Example:

```
Cardinal.continue('cca',
{
        AcsUrl:"https://0merchantacsstag.cardinalcommerce.com/Merch
        antACSWeb/creq.jsp",
        Payload:"eyJtZXNzYWdlVHlwZSI6IkNSZXEiLCJtZXNzYWdlVmVyc2lvbiI6
        IjIuMi4wIiwidGhyZWVEU1NlcnZlclRyYW5zSUQiOiI1YmJjZjRkYS1iZWU5LT
        QwNWUtYjE2NS04ZjEzMWQ0NjYxZTkiLCJhY3NUcmFuc0lEIjoiNWVlY2Zl
        Y2YtODUwYy00MGEyLTgyOGYtZjA5MGE4NWM1ZWRkIiwiY2hhbGxlbm
        dlV2luZG93U2l6ZSI6IjAyIn0"
 },
{
OrderDetails: {
                TransactionId: "J10RCDJbUOLW7St1j9O0"
        }
});
```

14. Handling the CCA Response

- Songbird.js will handle all the user interactions until CCA has returned back the authentication result. The next step in the integration is to add logic to your **payments.validated** event to handle specific return values for CCA.
- You will receive two elements in the response: **data, JWT**
- The field **data.ActionCode** should be used as the primary transaction status indicator.
- Below are the possible values for ActionCode and what they indicate:
    - **NOACTION** - Authentication was not applicable.
    - **SUCCESS** - Authentication was completed successfully.
    - **FAILURE** - Authentication resulted in a failure.
    - **ERROR** - An error was encountered.

For more details :
https://cardinaldocs.atlassian.net/wiki/spaces/CC/pages/98315/Response+Objects

JSON Response Example:


```
{
        "iss": "<API Identifier>",
        "iat": 1577091255,
        "exp": 1577098455,
        "jti": "61729408-0c2a-4be6-8ee2-892b4d05ce00",
        "ConsumerSessionId":
       "1_7076814d-5117-4940-9e4e-aa92a8b03c01",
        "ReferenceId": "1_7076814d-5117-4940-9e4e-aa92a8b03c01",
        "aud": "19ec6910-5048-11e6-8c35-8789b865ff15",
        "Payload": {
         "Validated": true,
         "Payment": {
           "Type": "CCA",
           "ProcessorTransactionId": "J10RCDJbUOLW7St1j9O0",
           "ExtendedData": {
           "CAVV": "MTIzNDU2Nzg5MDEyMzQ1Njc4OTA=",
           "ECIFlag": "02",
           "ThreeDSVersion": "2.2.0", // Possible values: 1.0, 2.0, 1.0.2, 2.1.0, 2.2.0
           "PAResStatus": "Y",
           "SignatureVerification": "Y"
         }
        },
```

```
            "ActionCode": "SUCCESS",
            "ErrorNumber": 0,
            "ErrorDescription": "Success"
           }
}
```


JWT Response Example:


```
eyJhbGciOiJIUzI1NiJ9.eyJpc3MiOiI1ZGJhYmEzY2FmYTgwZDFkOThmNzhmZjgi
LCJpYXQiOjE1NzcwOTEyNTUsImV4cCI6MTU3NzA5ODQ1NSwianRpIjoiNjE3Mjk
0MDgtMGMyYS00YmU2LThlZTItODkyYjRkMDVjZTAwIiwiQ29uc3VtZXJTZXNza
W9uSWQiOiIxXzcwNzY4MTRkLTUxMTctNDk0MC05ZTRlLWFhOTJhOGIwM2Mw
MSIsIlJlZmVyZW5jZUlkIjoiMV83MDc2ODE0ZC01MTE3LTQ5NDAtOWU0ZS1hYTk
yYThiMDNjMDEiLCJhdWQiOiIxOWVjNjkxMC01MDQ4LTExZTYtOGMzNS04Nzg5Y
jg2NWZmMTUiLCJQYXIsb2FkIjp7IlZhbGlkYXRlZCI6dHJ1ZSwiUGF5bWVudCI6e
yJUeXBlIjoiQ0NBIiwiUHJvY2Vzc29yVHJhbnNhY3Rpb25JZCI6IkoxMFJDREpiVU
9MVzdTdDFqOU8wIiwiRXh0ZW5kZWREYXRhIjp7IkNBVlYiOiJNVEI6TkRVMk56Zz
VNREV5TXpRMU5qYzRPVEFcdTAwM2QiLCJFQ0lGbGFnIjoiMDIiLCJUaHJlZURT
VmVyc2lvbiI6IjIuMi4wIiwiUEFSZXNTdGF0dXMiOiJZIiwiU2lnbmF0dXJlVmVya
WZpY2F0aW9uIjoiWSJ9fSwiQWN0aW9uQ29kZSI6IlNVQ0NFU1MiLCJFcnJvck5
1bWJlciI6MCwiRXJyb3JEZXNjcmlwdGlvbiI6IlN1Y2Nlc3MifX0.0USHnSBp6FMZO
Z7qaaX3DXZCVVgwJLUMEGUQ5NX0zVE
```


15. JWT Validation (Optional)

- Once the response JWT is received in the payments.validated event, you can verify JWT and extract the results.
- **Note**: JWT validation should only ever be done on the server side for security reasons.
- You can validate JWT by JAVA, .NET, or PHP programs in the backend.
- JWT validation is optional.

For more information and code samples:
https://cardinaldocs.atlassian.net/wiki/spaces/CC/pages/360593/JWT+Validation

16. Authorisation

- After the successful validation of JWT / CCA response, you need to call **Apexx 3DS Verify Authentication API** with the three_ds authentication data.
- After this call you will receive a final payment response with the SUCCESS or FAILED status.

For API request/response:
https://sandmgw.apexxfintech.com/mgw/v2/api/doc#operation/threeDSVerifyAuthenticationUsingPOST

Request Example: (Not all the fields are required, its required based on gateway)

```
{
        "_id": "string",
        “paRes”: “string”,
        "three_ds_authentication": {
            "xid": "string",
            "cavv": "string",
            "cavv_algorithm": "string",
            "3ds_status": "string",
            "eci": "string",
            "3ds_enrolled": "string",
            "directory_server_transaction_id": "string",
            "3ds_transaction_status_reason": "string",
            "3ds_score": "string",
            "3ds_preference": "string",
            "3ds_mode": "string",
            "3ds_version": "string",   // Possible values: 1.0, 2.0
            "3ds_result": "string",
            "3ds_server_transaction_id": "string"
        }
}
```

- Conclusion: Based on above all steps you can integrate 3DS 2.0 with CSE.

Sample Code of implementation:

- You can see all the steps into below link : **https://drive.google.com/file/d/1iYMChwMX_ZDSnbjYHY9T774zUi8-8NuS/view?usp=sharing**

3DS Verify Authentication API request field mapping:

Authentication API	JWT fields / value	Received at	3DS Version	Comment
cavv	Payload.Payment.ExtendedData.CAVV	Cardinal Response	1.0, 2.0	
cavv_algorithm		Generated	2	

		by merchant		
3ds_status	Payload.Payment.ExtendedData.PAResStatus	Cardinal Response	1.0, 2.0	
eci	Payload.Payment.ExtendedData.ECIFlag	Cardinal Response	1.0, 2.0	
3ds_enrolled	if three_ds_enrollment = true then value should be "Y" otherwise "N"	3DS Card Enrol	1.0, 2.0	
directory_server_transaction_id	three_ds.directory_server_transaction_id	3DS Card Enrol	2.0	
3ds_transaction_status_reason				Not required
3ds_score				Required only for CB transactions, not received
3ds_preference				Merchant choice which transactions they want to continue process for authorisation
3ds_mode				Optional
3ds_version	1.0 or 2.0	3DS Card Enrol		
3ds_result				Not required
3ds_server_transaction_id	Payload.Payment.ProcessorTransactionId	Cardinal Response	1.0, 2.0	

3DSv1 Fallback

When processing 3DS version 2 you are specifying in the request that you want the payment to go through 3DS version 2.X (X being the highest available authentication method i.e. 2.2) as shown below.

```
"three_ds": {
        "three_ds_required": true,
        "three_ds_version": "2.0",
        "reference_id": "0_2f97e49b-c85f-4f52-b8da-a105408b39ea"
    }
```

Though in the request you are requesting that the payment goes through 3DS v2 there are situations where it might not be possible to process the transaction

using the 3DS v2 protocols. In this situation you will receive a response to your initial request that indicates you need to process the payment via the traditional 3DS v1 payment flow. This is known as 3DS v1 fallback. The response that you receive in the 3DS v1 fallback scenario is as below.


```
{
   "reason_code": "0",
   "_id": "174fac32ed2546968fcb940f5dc3a410",
   "three_ds": {
       "three_ds_required": "true",
       "three_ds_version": "1.0.2",
       "specification_version": "1.0.2",
       "acsURL":
"https://merchantacsstag.cardinalcommerce.com/MerchantACSWeb/pareq.jsp?vaa=b&gold=A
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAA",
       "paReq":
"eNpVUstuwjAQvPsrEOo5jo3FS4ulFNQWShFteVTcgrNq0kISnATC39dOoLQ+7ex61zOzhkWoEUfvqAqNEl
4wy/xPbETBoLkZJvE6mu/G56x4XT4j/wrdpoS594YHCUfUWZTEkjmuw4FeITEjtAr9OJfgq8P9eCZFu8dZG
```

```
+gFEtijHo9kSzDBueBufYDWaQKxv0fppViWjYcozlGFjWkeOECrAgGVFHGuz7LLzdArIFDonQzzPO1Tejqd
HHXeGkpJoRU6KtkDtXUC9EZvXtgoM6rLKJDr79lxuwvS5WM5wY8ZW65W081icvBH3gCovUEg8HOU3OXM7bB
Ww+30W52+YECrPAF/b8nIOyYct2v01pBAah/yasSELf3NGEGF1hgro0gYG34RASzTJEZzxxj8GxsNN+bDJ2
uzyo1xoit6osuszxWu2iNjDees7o8qn6jtoZct0svCTfTvI/wAyo+tRQ==",
        "three_ds_enrollment": true,
        "acq_id": "WkNvbldpUGxJeXN1UVVLZTJqaDA=",
        "authentication_transaction_id": "ZConWiPlIysuQUKe2jh0",
        "acsURL_http_method": "POST",
        "reference_id": "0_2f97e49b-c85f-4f52-b8da-a105408b39ea"
    },
    "created_at": "1626161867990"
}
```


In this situation you need to use the value "three_ds_version": "1.0.2" to identify that the transaction has fallen back to 3DS v1. Where this is the case you need to initiate the standard 3DS v1 payment flow as outlined here. For 3DS v1 you redirect the customer to the ACS page to complete authentication providing the details outlined in the guide mentioned above to complete that payment flow.

Causes

Causes of 3DS v1 fallback could be related to the card issuer not being enrolled for use of 3DS v2, problems with the device data collection, or issues with the 3DS v2 server not allowing the transaction to be authenticated using this protocol.

Testing

In order to test the 3DS v1 fallback you can send a request for 3DS v2 but use a card number that is enrolled for 3DS v1 (for example, 4000000000000002).

Verify Authentication

Sample verify authentication request for a 3DS v1 fallback can be found below.


```
{
    "_id": "41fa3e72b31d433db9780a1ddb8f7665",
```

```
    "paRes":
"eNrVmFmzqsiygN+N8D/s6PPo6QYUETrcK6KYUZlBhjcEZBYEGeTX33KtPXWffTr2vW93RaywSLOyMsnKr9
LaW2kbx6wZh30bv+3luOuCJP6URZ9/2xS7FDtxk74TgT7FPTuR6G9vew0Ycfeu8D7ajbF0t8I0ZuLeH8lAW
823srwx+KH7DLWHuO2y+vaG/YH+sd4jXx/hQm2YBrfH2z4I77SkvOEEtcaIPfL1cV/FrcS+RfE16MvHHvl4
3CPf52n9a9RBp6csepPX3SZgedlwU0NBU9p20zKyDVxlwec98tLYR8Ejfluj6A4l1ttPGP4niv2JknvkXb5
vXuZAVffQNrZBsT3yo2QPX08b38LnG4Fv9si3p308NfUthhowuG/jPfLduSa4vaF//YOqL+nect/2j6z60S
nq5RS22yPv8n33CB599+btkS+jfRgMwxsAgAYOX1Y6eA1zw3F8juMTfl7BYN9V9nGYvaFb6BT8fJ8FyqRus
0davVz9q2CPvFxB3vP5tjez5AYXa+NPU1XeYBbTx6P5E0HGcfxj3PxRtwkCHUYRlEKgQtRlyb9++5gVR9Lt
Wv+vpjHBrb5lYVBmc/CAO0OOH2kdffrm28/MWMbLEoYYHPM7NPV7iOG3318SdINtoU3k50Z/iOxXVvm7s20
X/N6lAfZa4G+G3vZGfI1fOyL+ZBvS59/+9SulwWZJ3D3+L658deNHC1/tnYOyj98c0970h/RkWWY6tTsEsR
pdAy2OIPbnr/M+NPfIN9+/BPaRxR/e1odiJo6caUQ2jadu3NPBTD+PrGxrXuyG6vmWihXxKFm69s+cYeFO6
HdU2F3AlW/Cez0QJ5N8doUu3peLBCWbQMCuIpizlaRe07yLTLEukm48dGxalkbq9s8DplX5qkByYDmnADnV
cW9NTbC+k5fdRGsCTS8XAdVzosd6esRRckKMptNtRPwo16jIKKdGwjb5LmRRQASzeFIpLqjbUhACIumYrBN
pppGQjDyM6HJx9ASIntXMnzXE7tXnKh6jrYrBmpikInsIPo+bT4ruB2O4MwBgO4TezRg9INllHFhdrjdnM9
7kT2jp6dt3cqPUnb9TjzNWVwdxXR83YShno27KFWFZOt8IvG5NXJLqylF4bMHnzz/sqi+ZOcbPj0y4W5Rig
0fwMTL7Sx6HDyWAhGCUz0zQRtktKD+xcVV/ek9g++9Pqv39C8CYUPD9makriNMw/ven02cZ8qqG+qb1WU2z
+t+fmM+2uUf+vs77wkzcPrIrrCvIS1mSuFPOMGAKEjBKNEikA1DVEA/8qbMUIANUYMy7YEqXDatzNDPaQBa
8iZnBgU6UMw08C5Rna7mQdXRkRo8967rEjYYWCOUgG/gognfZkZvOrOceisDZphdOGS4OVoaVUkpwHN7AxO
VAf1lcLmjQWYx1OPjCuY+EsgoszpZpSQCYzYFJPp2F8xwx9Bw5UqKj3CimoSLn8JucW8usPb3+HSiDPuXcX
6U5jch6NzL6u08CNx7O9swpMujerTO0zFnrtLnMnPaK/CWjJ1lYLizn5Qv1lI16FL5FxH+PiKHZyxp7+O4h
9y3uLNPyh8VJlvU1//SFsvecMVkudFuxZAMdhY93xXKT8aMVznewQeKVxl9vy/CVGVMaWd07HGtfSodQATA
LtA5YWH8JpwEWaug1A8c00PSIbo17MgqsW5DxlumUrrlvihybYT0XSLWx+i4EanqcdAz05GkbjObaXC4IAe
FktDsRGBXtdpVilWeDu1bdQTau0onVkduxwJLBn4s431CrDdm4zd3PN2YUzx4RGoSB+WmbGMvFIDk1d2gTS
jOjrCpcdujkvtB3m80gXdv5iQxXslHhWc2R610bM+YmjfDh8qCtOT34uyd23V49li5gdAZzC8n8st518z0S
a9bbSjJRBHYdEm10q437OhGvXdb07EFduc/xxkmBorbpxiTB3dtVVZbjD/W2XAhVzcXHSqWFO6uY7qHe8Qd
1fSgPsatUnXwet6hIiXiteqJ782uN6eotLp9oRO8sBE0s0aVvOzVklgswcgAEKj1ZDD3mHxUgGjIHcgBkmh
MYhxFMIAgrDlaSrnOwoviaBTOdlElaJHQ6pDBbKLdcHFKvsDZ0eSkVCxoTfQdmu+JR39ETfT018BmFlVGGN
6PxqzL3XKOU6fcdGbGJ7tC0mc6XzWG5MKxzDlabihWxktUpcyXqySMH13ffTJkTWOAktHmVKBm3qIunVcd1
EyAAN9WcvokZygL1o+5EnaTBlYQbSWboGIziq0oMtKRpb+SBJx1HD+48W4RRCQwDK0a3eXqUub8zYrl4UQL
gr3cAjMZODyC1B3RYkWahuTfnulnnw07cdOIl6bmYPuLJbYM7PlWlnnB18o13Rc3aN9ZwPx1FVVljsV5eeK
E/8p3T+iohyBTZTFlwOAk3OkOIi/Qo8KtzGQbDjdXC3SgYmbGjKWpZopKFG22vywWJVw4jVo9HvEqpij/ZN
VEQjSMbFGZLnq+WW7Kzm1qAFbarTyXr7vptzXtycZ3j6yNMzY5Ct9ZKWi5MpncVRayH43Y8lRRL9joO+ERP
We5ZYYm6okO3G3v20gdzyYrT0Gt3a1uszttQ3HWegxn49linkE9UuEZWpFtq4ZBe0Ry7u8++DSSjnVL/fL0
0O37VJqUW++plt/N2qHp6AJXcDUeJdzcZKn8A/kee/wzw7DBDSNzjr4CngQUUOinuaZEJ1IjCNHU8YGn8n9
AIk/pf4Pgdjb8G++XiR9z/BPbf4AmRLn+BvXlZU6+CoGUad1lLGmVLmpYLJQeoPEM9rH5JcZmVJ9X6Jhv9n
```

```
DvJoPjwNZWZ81meWAucPlZOLJqj6sAhoSUxB+GHtLZ4W0nDymguFd/pa+pxcc5Pb23DwqSeIcuZMg2+wv0g
F2XxKtbA8RvZ0Mflgku+gH182L6roJKoYF5Gs7r132H+Hyg/CtvlgkW53Q6Z3B2RNnxkWKGrXc2uMDaJ8hh
LD9kNaY7Q6lCtTtzqUoXIFlcdpfWV+wU2X0/5gliK9XwS8Ei1ymu2UnVhsIWw9coi2T63ZWrgPUiAMzgByx
VafE64GrkfA4uIpWP9BI3Z4Sl6tl2MIa7PWljxy8Vl3WVw10R1SylY1uiGH7SG2pFGCYPxAqUA9v2apKN8i
8YqJA/IrpFPlRbdx2YeiyJpelvVD8d5uUCO5g5DjoyuremDrgpIyJ2MTZMktNUnRIRKGZLxq0dIIQUu3pWq
luZwR1yC4JJfe+HE4rNeiOYDWy7sjOwLaxRnd73mOPnamuvzpb0XPK3j6X3NS5JLWSavO0BUStNFsChkPlC
uVLKgfwWdCt8T7XH8gczROdWwR7Ly2qBsSTaVCFkI45Mu0+Q7gqVR92Q6ADxPHvguFzbHvEko+th3F1U4Eu
1y0a467UvTEnEjRyOjLsnAewEWZhmMbPK+pgZ0EYEFyIJEEH7WbMFD5x2kzJn32eh28dQ5NQJzqGikyP0zY
63le3zTKOOKTIkVXuQV/RzijdjV0bie6mMxn6bCKddHCHglxHdmOR6paepo8mwFYTshwAjOu42P+uL54FsF
ZSfKRhMtXJDOkKeOaO7uddZb0TO4EfJTyzARWnIFwcPjoqTPlJ612oaULhqR6v4MVpXsEwp/AIrkWgmprY4
n9+mqcR5pvSZQJUaQ7iV/HCsYUA7bISkR0XmLY+J9vlJZuGHctUCeEHm2kTR03GcxeVegVrhDJ45ByIba6s
lhTciNrGh8zuZz1+zuBoBNzBmd+MINJkUZByTok8t0GEkvBW5xXh9twkhPinENczpEnHu2PgVE6OYnmbdLf
zr/IkinVz92J753ygci5i0ebQlt+9NOWUx+tVPWyRcYXjKRG33lhZTLRpr+CUuwGH8ZTPbIfe03x+kDS/xh
uGz09w4a4u8vPbSMKvDz1UO/gzTn0G+y/P/JMSFJtJR/HHPLxfeDjuMBUGEBkuClwSRHOObA8GzqcbJiY/N
wHcPeDdsVgd94f3tzmrNi3MvTxW6ukwajO/qZuaWw7WlUtneda+dpyOrE5cljTyXd2c34m6Lp0hU/OYqz9v
VdbmZltUWf/LbZzH5/PmGcbkFACHWQYT554A5Hws0ezEquUobzrp5UHRH1Xqy8iGtinLr0u+u5jccMzjtig
h50ylUUs6u2g93KoML95F8mlKlv9JBhLNcxcp4Q+U4tXeRh+tZze91MR9GJqUDCIHAwRoL48a+y65xP2eBX
fdUp0TapvHy56DcyGtFRH/JNvX1Q/nFCmHwl30Kizl0JrN172xpIBI5bj9TEpnUw2NV4U6HxpOwP1hAdBL3
K5Wa5QM/n1ZA4W9YIlMMosXBNusbk0bZA9N6Z6jgHWybbHQsHHlThXa3gaXcnpg732ns/ECM5inDHwrI20J
ymk5GvwT9qs+M7UK0PoNojeEF3/t7owG7ra6vzvgM4IOIjdbIune11tN/N9l0hZC7pUd9Ubqh/VrtzTSGG4
QL0ITwZ5Ma3uHRIPBtGd4/bOUsbwdLM9qk3+EySJK3dxu26G2XEOVzX9ycfjxGhdXbhzl7vDHHWqXxfrFVe
M4y0utwiFv7K2SHaafQ23TFtlNCrowfCbHekmxr6Wgo1jx6O16u6nQ63Wr+Sd/5YBklfiNlj5FH0rkgx2fb
BKZWp5WImSNHZdYpFHoFzUmX28eBv+W6ONoRsiR7Q5AHuqNw0eA8NfQOvIvj79SCOfXyn7Ba7mhw6JffX78
rcmhXlSCkXT7GISCTFNXZBUIK+K9H68qjiICImDeH7jPNxr/Gea9NTEMdeOXZYs+wFf7hrnBlfNyH/AVXk+
xUI8u1a5PuFyfvF8PvN9esu88cb7f8B0eFlwg==",
    "specification_version": "1.0.2"
}
```

Flow Comparison

Normal 3DS 2.0 flow:

1. Process Enrolment(payment/direct) request with three_ds_required = true, three_ds_version = 2.0 and reference_id.
2. APEXX will return a successful response with three_ds_version 2.1.0/2.2.0, specification_version = 2.1.0/2.2.0, acsURL, authentication_transaction_id, directory_server_transaction_id, reference_id, and paReq etc.
3. Call the ACS URL by JS with paReq.
4. Popup will display on the same page.
5. Complete the challenge.

After receiving successful response from Cardinal, call Verify Auth API request as below:

```
{
  "_id":".....",
  "three_ds_authentication":{
  "cavv": ".....",
   "3ds_status": ".....",
   "eci": ".....",
   "3ds_enrolled": ".....",
   "directory_server_transaction_id": ".....",
   "3ds_version": "2.0",
   "3ds_server_transaction_id": "....." }
}
```

6. APEXX will return a successful payment response.

Fallback to 3DS 1.0 flow:

1. Process Enrolment(payment/direct) request with three_ds_required = true, three_ds_version = 2.0 and reference_id.
2. APEXX will return a successful response with three_ds_version =1.0.2, specification_version = 1.0.2, acsURL, acq_id and paReq etc.
3. Redirect the user to the ACS URL with the paReq.
4. Challenge will display on the ACS server page.
5. Complete the challenge.

6. After completing the challenge, you will receive a response on the term URL from the ACS server with the paRes.
7. After receiving successful response from the ACS server, call Verify Auth API request as below:

```
{
  "_id":".....",
  "paRes":".....",
  "specification_version": "1.0.2"
}
```

8. APEXX will return a successful payment response.

Frictionless 3DS 2.0 flow:

1. Process Enrollment(payment/direct) request with three_ds_required = true, three_ds_version = 2.0 and reference_id.
2. APEXX will return a successful payment response with three_ds_version =1.0.2/2.1.0/2.2.0,
 three_ds_enrollment, authentication_result, eci, xid, cavv, pares, veres etc.