



M13: Implementing Payer Authentication Direct REST API

Version	1.1
Date	January 2021

1 Version Control

Revision	Date	Description
1.0	November 2020	Initial release
1.1	Jan 2020	Clarified use of Flex Microform transient token.

2 Table of Contents

M13: Implementing Payer Authentication Direct REST API	1
1 Version Control	2
2 Table of Contents	3
3 Introduction	5
4 Payer Authentication 3DS 2.x Overview	6
4.1 Prerequisites	6
4.2 Overview	6
4.2.1 High level steps	7
5 Payer Authentication 3DS 2.x Detailed Steps	9
5.1 STEP 1 - Setup Payer Auth API Request	9
5.1.1 Key Request Fields	9
5.1.2 Request Example 1 - Setup Payer Auth Service using FLEX token	9
5.1.3 Request Example 2 - Setup Payer Auth Service using card details	9
5.1.4 Request Example 3 - Setup Payer Auth Service using TMS token	9
5.1.5 Key Response fields	10
5.1.6 Reply Example 1 - Setup Payer Auth	10
5.2 STEP 2 - Device Data Collection	10
5.3 STEP 3 - Add event Listener	11
5.4 STEP 4 - Check Payer Auth Enrollment + Authorization API request	11
5.4.1 Key Request fields	12
5.4.2 Request Example 1 - Process a Payment + CONSUMER_AUTHENTICATION using FLEX token	12
5.4.3 Request Example 2 - Check Payer Auth Enrollment using Card Details	13
5.4.4 Request Example 3 - Check Payer Auth Enrollment using FLEX token	14
5.4.5 Key Response fields	14
5.4.6 Reply Example 1 - Process a Payment - Cardholder Enrolled	15
5.4.7 Reply Example 2 - Process a Payment - Cardholder NOT enrolled	16
5.5 STEP 5 - POST to stepUpUrl	17
5.6 STEP 6 - Receive the Authentication Result	18
5.7 STEP 7 - Validate Authentication Results + Authorization API request	18
5.7.1 Key Request fields	19
5.7.2 Request Example 1 - Process a Payment + VALIDATE_CONSUMER_AUTHENTICATION using FLEX token ...	19
5.7.3 Request Example 2 - Validate Authentication Results	20

5.7.4	Key Response fields	20
5.7.5	Reply Example 1 - Process a Payment + VALIDATE_CONSUMER_AUTHENTICATION	21
5.7.6	Reply Example 2 - Process a Payment + VALIDATE_CONSUMER_AUTHENTICATION	22
5.7.7	Reply Example 3 - Process a Payment + VALIDATE_CONSUMER_AUTHENTICATION	23
5.7.8	Reply Example 4 - Validate Authentication Results - Successful Authentication.....	24
6	Flex Microform with Payer Authentication	25
7	Payer Authentication Testing	25
8	Generating a REST API key	25
Appendix A - Extracting JTI from Flex Microform Transient Token.....		29
Disclaimer		29

3 Introduction

This document describes in detail how to implement Direct Payer Authentication, to support 3D Secure 2.x for card payment transactions on both mobile and website channels.

The Direct Connection API Payer Authentication workflow is an alternative to the Hybrid integration. It does NOT require the use of Cardinal Commerce's Songbird JavaScript library (and any other 3rd party libraries). Also, unlike the Hybrid model, it does not require knowledge of the Cardinal Commerce Payer Auth credentials. The Direct API can be used with the SOAP Toolkit API service or REST API for the backend services, but for new integrations, REST is recommended. This document will concentrate on REST.

In 1999 3D-Secure 1.0 (Payer Authentication service) was introduced to help prevent fraud for online purchases. The following card schemes support 3DS 1.0:

- Visa Verified by Visa,
- MasterCard and Maestro SecureCode,
- American Express SafeKey,
- JCB J/Secure,
- Diners Club ProtectBuy,
- Discover ProtectBuy.

Payer Authentication (PA) provides an additional level of security for online payments, whereby cardholders who are enrolled into PA provide a password to complete online transactions.

In 2020 3D-Secure 2.x is being rolled out. 3DS 2.x eliminates all the disadvantages of the 3DS 1.0. It collects ten times more data during the process, allowing the issuer to better determine if the customer is risky or not. It provides much better customer experience for mobile device users, and makes the whole PA process more flexible.

4 Payer Authentication 3DS 2.x Overview

4.1 Prerequisites

Before you start running 3DS 2.x transactions in Test or in Live environment, four initial requirements must be fulfilled:

- 1) Your Smartpay Fuse MID must be configured specifically for 3DS 2.x.
- 2) Your Smartpay Fuse MID must be setup with three credentials, provided by Cardinal Commerce. These will be visible in the Enterprise Business Center when set up.

IMPORTANT - Although not required in your integration, these credentials must be present for 3DS Direct to work correctly.

Parameter name	Description	Example
API Identifier	A non-secure value that should be passed within the JWT under the iss claim.	5b30e68e6fe3d126b4037b02
Org Unit Id	A non-secure value that should be passed within the JWT under the OrgUnitId claim.	5b2d8dacff626b051cb9f891
API Key	A secure value that should never be rendered or displayed anywhere within a browser/mobile app. This value must only ever be stored on the merchant server. The API Key is used to sign the JWT initialize the connection to Cardinal Commerce, and to verify the JWT signature received back from Cardinal. It must never be included within the JWT itself.	2725b3a8-af13-4e5b-ba77-c593a1207996

Contact the Support team to ensure that the gateway MID is configured for 3DS2 and the above credentials are in place.

- 3) You must have set up a REST security Key for your Transacting Gateway MID. See Section 8. Generating a REST API key for details.
- 4) Your customer browser version must be higher than in the table below:

Browser type	Version
Desktop	IE 10+ Edge 13+ Firefox 42+ Chrome 48+ Safari 7.1+
Mobile	iOS Safari 7.1+ Android Browser 4.4+ Chrome Mobile 48+

4.2 Overview

The following diagram gives a high-level view of the workflow, involved in the Direct API integration. Detailed descriptions of each step follow later in this document.

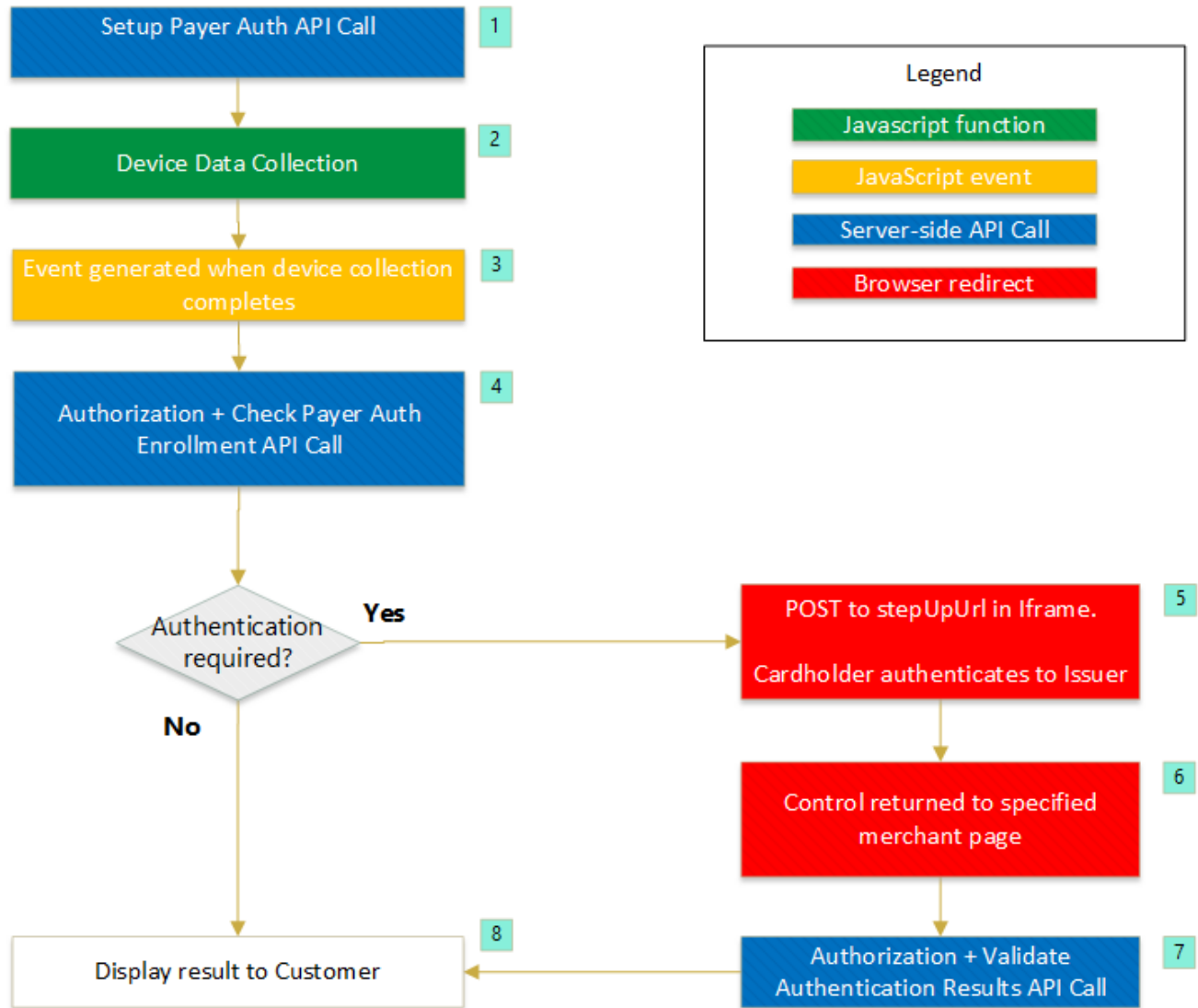


Figure 1 - High-level Workflow`

4.2.1 High level steps

Step	Description
1	Setup Payer Auth API call Call out to server-side to generate Setup Payer Auth API call with Card details/Transient Token. This call returns a deviceCollectionUrl and an accessToken JWT, which are required for Device Data collection.
2	Device Data Collection “Silently” POST the accessToken JWT to the deviceCollectionUrl returned by Setup Payer Auth (STEP 1) An event will be triggered when device collection is complete.

3	<p>Add Event Listener <i>Add a function to handle the device-collection complete event.</i></p> <p>profile.complete event A sessionID will be returned in the event payload, which is then used in the Check Payer Auth Enrollment API.</p>
4	<p>Check Payer Auth Enrolment + Optional Authorization API Call Call the Check Payer Auth Enrollment API to check whether the cardholder is enrolled into 3DS.</p> <p>Supply sessionID in the consumerAuthenticationInformation:referenceId field and specify consumerAuthenticationInformation:returnUrl to redirect the cardholder to in the event that authentication window is required.</p> <p>It is recommended to perform an authorization request in the same call. If the cardholder is NOT enrolled then the Authorization request will be processed, and the system will populate the request with the correct values to indicate that 3DS was attempted. If the cardholder IS enrolled and authentication is required then the Authorization request will be ignored.</p> <p>If authentication is required, then stepUpUrl and accessToken JWT will be returned.</p>
5	<p>POST to stepUpUrl POST the accessToken JWT to the stepUpUrl returned by Payer Auth Enrolment. Typically, this would be in an Iframe.</p> <p>The cardholder authenticates to the Issuer in a 3DS window, although in some cases this is not required.</p>
6	<p>Redirect to return Url The browser/Iframe is redirected to the returnUrl specified in the Payer Auth Enrolment check. authenticationTransactionId will be returned in the payload, and is used in the next step.</p>
7	<p>Validate Authentication Results + Optional Authorization API Call Call the Payer Auth Validate Service to retrieve various fields required in the Authorization request to indicate that 3DS was attempted and successful.</p> <p>Supply authenticationTransactionId in the consumerAuthenticationInformation:authenticationTransactionId field.</p> <p>It is recommended to perform an Authorization request in the same call, as the system will populate the Authorization request with the correct values from the Validation request. For this reason you must supply full order/billing details, as well as cards details, a FLEX transient token or TMS permanent token.</p>
8	<p>Display result to Cardholder</p>

5 Payer Authentication 3DS 2.x Detailed Steps

5.1 STEP 1 - Setup Payer Auth API Request

The Payer Authentication Setup service is called on the server side after the customer has entered their card details. Typically, you would use FLEX Microform to capture the card details, or you may have previously captured card details stored in a TMS token, but you can also supply “actual” card details. The Payer Authentication Setup service must be called on its own without including other Smartpay Fuse services.

5.1.1 Key Request Fields

See the Developer Center for comprehensive request and response details: https://developer.cybersource.com/api-reference-assets/index.html#payer-authentication_payer-authentication_setup-payer-auth

5.1.2 Request Example 1 - Setup Payer Auth Service using FLEX token

```
{
  "clientReferenceInformation": {
    "code": "cybs_test"
  },
  "tokenInformation": {
    "transientToken": "1E3HQL26RVJPQBP6ELASS2FRJ51A7F3KABU79H26NEE7C1Q3K3C5F993DF1042F"
  }
}
```

N.B The above example assumes that the expiry date fields have been included in the FLEX token.

IMPORTANT NOTE

The value of “**tokenInformation->transientToken**” is the value of the JTI claim in the JWT returned by Flex Microform. See **Appendix A - Extracting JTI from Flex Microform Transient Token** for further details.

5.1.3 Request Example 2 - Setup Payer Auth Service using card details

```
{
  "clientReferenceInformation": {
    "code": "cybs_test"
  },
  "paymentInformation": {
    "card": {
      "type": "001",
      "expirationMonth": "12",
      "expirationYear": "2020",
      "number": "400000xxxxxx0002"
    }
  }
}
```

5.1.4 Request Example 3 - Setup Payer Auth Service using TMS token

```
{
  "clientReferenceInformation": {
    "code": "cybs_test"
  },
  "paymentInformation": {
    "customer": {
```

```

    "id": "B103500B4BE49012E05341588E0A7655"
  }
}

```

5.1.5 Key Response fields

The Setup Payer Auth API will return the following key fields:

Field	Meaning
status	Indicates status of the call. Possible values: COMPLETED: the call was successful. FAILED: an error occurred.
consumerAuthenticationInformation: accessToken	JSON Web Token (JWT) used to authenticate the consumer with the authentication provider. Required for next step - Device Data Collection.
consumerAuthenticationInformation: deviceDataCollectionUrl	The deviceDataCollectionUrl is the location to send the Authentication JWT when invoking the Device Data collection process. Required for next step - Device Data Collection.
consumerAuthenticationInformation: referenceld	This identifier represents the device data collection session. Required for subsequent step - Payer Auth Enrollment Check.

5.1.6 Reply Example 1 - Setup Payer Auth

The following example illustrates the Setup Payer Auth REST response:

```

{
  "consumerAuthenticationInformation": {
    "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJqdGkiOiIwNTAzMwY0NS02NjE2LTRjYzYtODAYz1iNTRkZTg3MWNkMDkiLCJpYXQiOiJlMjMDQzZmVjczNDUzImV4IjE6IjVkdGZyYmYwMGU0MjNkMTQ5OGRjYmFjYSIsImV4cCI6MTYwNDMzMdk0NSwiT3JnVW5pdElkIjo1NWV1NzVmYThjYTYyMjAwZjdmOGI4MGY0IiwiaWF0IjoiIjE4ZjYyIiwiaXN0ODM1LTRiYTYtOWUwYi1hZDaxYjU1NTljYTYifQ.W4ADdw2x77-KflxMoQebIFWT8A0Gyj7hoWuFRroL0Ho",
    "deviceDataCollectionUrl": "https://centinelapistag.cardinalcommerce.com/V1/Cruise/Collect",
    "referenceId": "8f12a282-1835-4ba6-9e0b-ad01b5559ca6",
    "token": "AxizbwSTR5tEP2ZV1YH7AAIBURwwcdgeABM4ZNJMvRixBOTATAAAUANA"
  },
  "id": "6043273449186077803003",
  "status": "COMPLETED",
  "submitTimeUtc": "2020-11-02T14:29:05Z"
}

```

5.2 STEP 2 - Device Data Collection

Device Data Collection is initiated on the front end after you receive the Setup Payer Auth reply as described above. A hidden 1x1pixel iframe is rendered to the browser in order to profile the customer device. The response depends on the card-issuing bank and can take 3 to 5 seconds. If you proceed with the Check Payer Auth Enrollment service (see next step) before a response is received, Smartpay Fuse will fall back to 3D Secure 1.0.

Initiate a form POST in a hidden iframe and send the **accessToken** JWT to the **deviceDataCollectionUrl** URL returned by Setup Payer Auth:

```
<iframe name="ddc-iframe" height="1" width="1" style="display: none;">
```

```
</iframe>
<form id="ddc-form" target="ddc-iframe" method="POST" action="<<deviceDataCollectionUrl>>">
<input type="hidden" name="JWT" value="<<accessToken>>" />
</form>
```

You can add JavaScript to invoke the iframe form POST automatically when the window loads - if you already have the response from Setup Payer Auth. Place the following JavaScript after the closing `</body>` element:

```
<script>
window.onload = function() {
var ddcForm = document.querySelector('#ddc-form');
if(ddcForm) // ddc form exists
ddcForm.submit();
}
</script>
```

You may also choose to submit the form at a different time, but you must do it before requesting the Check Payer Auth Enrollment.

5.3 STEP 3 - Add event Listener

After device data collection has completed, an event will be generated. You will need to trap this event. The Example below shows how to subscribe to the event and add JavaScript to receive the response from the device data collection iframe. Place the JavaScript after the closing `</body>` element.

Note that the event.origin URL is variable depending on your environment:

- Test: <https://centinelapistag.cardinalcommerce.com>

- Production: <https://centinelapi.cardinalcommerce.com>

```
<script>
window.addEventListener("message", (event) => {
  // {MessageType: "profile.completed", Session Id: "0_57f063fd-659a-
  // 4779-b45b-9e456fdb7935", Status: true}
  if (event.origin === "https://centinelapistag.cardinalcommerce.com") {
    let data = JSON.parse(event.data);
    console.log('Merchant received a message:', data);
  }
  if (data !== undefined && data.Status) {
    console.log('Songbird ran DF successfully');
  }
}, false);
</script>
```

N.B. There is no requirement to retain any of the data items received in the event payload, however **data.SessionId** will be the same value as **consumerAuthenticationInformation:referenceId** returned by Setup Payer Auth.

5.4 STEP 4 - Check Payer Auth Enrollment + Authorization API request

To check whether the cardholder is enrolled in 3DS, submit a **Check Payer Auth Enrollment** API call **OR** submit a **Process a Payment** request and include a Payer Auth Enrollment check (**CONSUMER_AUTHENTICATION**) in the same call. The latter approach is recommended in almost all circumstances, unless there is a specific reason to separate the Payer Authentication flow from the payment flow. With the combined call, if the cardholder is NOT enrolled, then the authorization request is run and automatically populated with the correct parameters to indicate that 3DS was

attempted by the merchant. If the cardholder IS enrolled then the Authorization request will NOT be run and must be submitted again after the authentication process has completed.

5.4.1 Key Request fields

It is recommended to include as much data as possible in your request, as this will increase the chance that the Issuer will not require a challenge screen. See the Developer Center for comprehensive request and response details:

- Process a Payment:
 - https://developer.cybersource.com/api-reference-assets/index.html#payments_payments_process-a-payment
- Check Payer Auth Enrollment:
 - https://developer.cybersource.com/api-reference-assets/index.html#payer-authentication_payer-authentication_check-payer-auth-enrollment_requestfielddescription

The following request fields are mandatory:

- **processingInformation:**
 - **actionList:** the static value **"CONSUMER_AUTHENTICATION"**
N.B Applicable only to the Process a Payment API.
- **Payment card details:** FLEX transient token, TMS token or card details.
- **consumerAuthenticationInformation :**
 - **returnUrl:** In the event that a StepUp challenge screen is required, Smartpay Fuse adds this return URL to the step-up JWT and returns it in the response of the Payer Authentication enrollment call. The merchant's return URL page serves as a listening URL. Once the bank session completes, the merchant receives a POST to their URL. This response contains the completed bank session's transactionId. The merchant's return page should capture the transaction ID and send it in the Payer Authentication validation call.
 - **referenceId:** the value of **consumerAuthenticationInformation:referenceId** returned by the Payer Auth Setup API.

5.4.2 Request Example 1 - Process a Payment + CONSUMER_AUTHENTICATION using FLEX token

```
{
  "request": {
    "clientReferenceInformation": {
      "code": "1604399973588"
    },
    "processingInformation": {
      "actionList": [
        "CONSUMER_AUTHENTICATION"
      ]
    },
    "orderInformation": {
      "amountDetails": {
        "totalAmount": "19.99",
        "currency": "GBP"
      }
    },
    "billTo": {
      "firstName": "John",
      "lastName": "Doe",
      "address1": "High House",
      "locality": "Redditch",
    }
  }
}
```

```

    "postalCode": "WR7 3DQ",
    "country": "GB",
    "email": "john@test.com"
  }
},
"tokenInformation": {
  "jti": "1C40V9WCUEZR3T0D3Z2TBAXJMJHF42K0C419SWAN60G0BUCK5TKH5FA136F1E1B6"
},
"consumerAuthenticationInformation": {
  "referenceId": "18818b40-05e0-40e9-a3d9-bf9bc6235ef8",
  "returnUrl": "http://example.com/redirect/"
}
}
}

```

N.B The above example assumes that the expiry date fields have been included in the FLEX token.

IMPORT NOTE

The value of **"tokenInformation->jti"** is the value of the JTI claim in the JWT returned by Flex Microform. See **Appendix A - Extracting JTI from Flex Microform Transient Token** for further details.

5.4.3 Request Example 2 - Check Payer Auth Enrollment using Card Details

```

{
  "clientReferenceInformation": {
    "code": "cybs_test"
  },
  "orderInformation": {
    "amountDetails": {
      "currency": "GBP",
      "totalAmount": "10.99"
    },
    "billTo": {
      "address1": "1 Market St",
      "address2": "Address 2",
      "administrativeArea": "CA",
      "country": "US",
      "locality": "san francisco",
      "firstName": "John",
      "lastName": "Doe",
      "phoneNumber": "4158880000",
      "email": "test@test.com",
      "postalCode": "94105"
    }
  },
  "paymentInformation": {
    "card": {
      "type": "001",
      "expirationMonth": "12",
      "expirationYear": "2025",
      "number": "4111111111111111"
    }
  },
  "consumerAuthenticationInformation": {
    "returnUrl": "https://example.com/",
    "referenceId": "12345678"
  }
}

```

5.4.4 Request Example 3 - Check Payer Auth Enrollment using FLEX token

```
{
  "clientReferenceInformation": {
    "code": "cybs_test"
  },
  "orderInformation": {
    "amountDetails": {
      "currency": "GBP",
      "totalAmount": "10.99"
    },
    "billTo": {
      "address1": "1 Market St",
      "address2": "Address 2",
      "administrativeArea": "CA",
      "country": "US",
      "locality": "san francisco",
      "firstName": "John",
      "lastName": "Doe",
      "phoneNumber": "4158880000",
      "email": "test@test.com",
      "postalCode": "94105"
    }
  },
  "tokenInformation": {
    "transientToken": "1E4T90Y6Y43QTIH9J0WQMZ6US1YKID1PJZO5S3CYHED43HTAPMUL5FB3EAC147B3"
  },
  "consumerAuthenticationInformation": {
    "returnUrl": "https://example.com/",
    "referenceId": "12345678"
  }
}
```

N.B The above example assumes that the expiry date fields have been included in the FLEX token.

5.4.5 Key Response fields

The **Process a Payment** API will return the following key fields:

Field	Meaning
status	Indicates whether a challenge screen is required. Possible values: <ul style="list-style-type: none">- PENDING_AUTHENTICATION: A challenge screen is required.- AUTHORIZED:- PARTIAL_AUTHORIZED:- AUTHORIZED_PENDING_REVIEW:- AUTHORIZED_RISK_DECLINED:- PENDING_REVIEW:- DECLINED: All the above values indicate that a challenge screen was not required and the authorization request was submitted.
consumerAuthenticationInformation: stepUpUrl	The fully qualified URL that the merchant uses to post a form to the cardholder in order to complete the Consumer Authentication transaction for the Cardinal Cruise API integration.

consumerAuthenticationInformation: accessToken	JSON Web Token (JWT) used to authenticate the consumer with the authentication provider. Note - Max Length of this field is 2048 characters.
---	---

The **Check Payer Auth Enrollment** API will return the following key fields:

Field	Meaning
status	Indicates whether a challenge screen is required. Possible values: - PENDING_AUTHENTICATION A challenge screen is required. - AUTHENTICATION_SUCCESSFUL The card is enrolled, but no further authentication is required. - AUTHENTICATION_FAILED The card is enrolled, but the payer could not be authenticated.- INVALID_REQUEST An error has occurred.
consumerAuthenticationInformation: stepUpUrl	The fully qualified URL that the merchant uses to post a form to the cardholder in order to complete the Consumer Authentication transaction for the Cardinal Cruise API integration.
consumerAuthenticationInformation: accessToken	JSON Web Token (JWT) used to authenticate the consumer with the authentication provider. Note - Max Length of this field is 2048 characters.

5.4.6 Reply Example 1 - Process a Payment - Cardholder Enrolled

The following example illustrates the **Process a Payment** response for a cardholder who IS enrolled in 3DS:

```
{
  "clientReferenceInformation": {
    "code": "1604405677017"
  },
  "consumerAuthenticationInformation": {
    "acsUrl": "https://0merchantacsstag.cardinalcommerce.com/MerchantACSWeb/creq.jsp",
    "challengeRequired": "N",
    "stepUpUrl": "https://centinelapistag.cardinalcommerce.com/V2/Cruise/StepUp",
    "authenticationTransactionId": "xclgZXLwa6r1Le8fEuS0",
    "pareq": "eyJtZXNzYWdlVHlwZSI6IkNSZXEiLCJtZXNzYWdlVmVyc2l2bviI6IjIuMS4wIiwidGhyZWVlbnNlcnZlclRyYW5zSUQiOiI5YTZiNWlxMS0zMTQ4LTQ3M2MmYTI3OS03YmM4NWNmNDgwODciLCJhY3NUcmFuc01EIjoInjZHYTI3OTctODQ4Ny00ZTkwLWlZzYmYtMDkyZmJmZkZkZDIiIiwY2hhbGxlbmdlV2luZG93U2l6ZSI6IjAyIn0",
    "directoryServerTransactionId": "26269f8b-2bbb-4bef-a4ab-ce2a548e75fd",
    "veresEnrolled": "Y",
    "threeDSSTransactionId": "9a6b5b11-3148-473c-a279-7bc85cf48087",
    "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJqdGkiOiJjNmNjNjNiMy1kMWM1LTQ2MWMuOTQ0MTQ1ZDZjMmU2ZWYiLCJpYXQiOiJlM2M0MDU2OTQsImIzcyI6IjVjVkdZdGZyYmYmGU0MjNkMTQ5OGRjYmFjYSIsImV4cCI6MTYwNDQwOTI5NCwiT3JnVW5pdElkIjoInWV1NzVmYThjYTYkYmJAwZjdm0GI4MGY0IiwiaWF0IjoiYmVhbnRBQ01NXZWIvY3JlcS5qc3AiLCJQYX1sb2FkIjoiaXNlKdFpYtTnpZV2R5Vkh5d1pTSTZJa05TWlhfFaUxDSnRwE56WVdkbFZtVn1jMmx2Ym1JNk1qSXVNUzR3SW13aWRHaH1aV1ZFVTF0bGnuWmxjbFJ5WVc1e1NVUWlPaUk1wVRaaU5XSXhNUzB6TVRRNExUUTNMk10WVRJm09TMDNZbU00T1d0bU5Ez3dPRGNpTENKaFkzTlVjBUZ1YzBsRU1qb2l0alp0wVRJM09UY3RPRFE0TnkMfPua3dMV016Wm1ZdE1Ea3labUppTxpCa1pEwm1JaXdpWTJoaGJHeGxiBWRsVjJsdVpHOTNVmMw2W1NjNk1qX1Jbja1LCJucmFuc01EIjoieGNsZ1pYTHdhNnJsTGU4ZkV1UzAifSwi2TjQzWn0aWZ5UGF5bG9hZCI6dHJ1ZSwiUmV0dXJvVXJsIjoiaHR0cDovL2xvY2FsaG9zdDo4MDE4L2N5YnMvM0RTMkRpcmVjdFJlc3QvcmlkaXJ1Y3QucGhwIn0.d7whZpLBFhQbfc0C8spxf8Lj-xZLGmM80BjGPY0sW1w",
    "specificationVersion": "2.1.0",
    "token": "AxjzBwSTR6YjyMUhH1XeAAIBT9s4wzW70hfzWTRdVSTL0YsQTKwDwAAAiBHJ",
    "acsTransactionId": "66aa2797-8487-4e90-b3ff-092fbb30dd6b"
  }
}
```

```

    "errorInformation": {
      "reason": "CONSUMER_AUTHENTICATION_REQUIRED",
      "message": "The cardholder is enrolled in Payer Authentication. Please authenticate the ca
rdholder before continuing with the transaction."
    },
    "id": "6044056945086494103006",
    "status": "PENDING_AUTHENTICATION",
    "submitTimeUtc": "2020-11-03T12:14:55Z"
  }
}

```

5.4.7 Reply Example 2 - Process a Payment - Cardholder NOT enrolled

The following example illustrates the response for a cardholder who is NOT enrolled in 3DS, where the Authorization request was approved by the issuer:

```

{
  "_links": {
    "void": {
      "method": "POST",
      "href": "/pts/v2/payments/6049548642236152503002/voids"
    },
    "self": {
      "method": "GET",
      "href": "/pts/v2/payments/6049548642236152503002"
    }
  },
  "clientReferenceInformation": {
    "code": "MOB12345"
  },
  "consumerAuthenticationInformation": {
    "authenticationTransactionId": "Y2wdSOau0hicHxtGPY00",
    "veresEnrolled": "U",
    "threeDSServerTransactionId": "66d867e6-68db-47a3-8bdf-b459f73fdc7a",
    "ecommerceIndicator": "vbv_failure",
    "directoryServerErrorCode": "101",
    "directoryServerErrorDescription": "Invalid Formatted Message Invalid Formatted Message",
    "specificationVersion": "2.1.0",
    "token": "Axj//wSTR/Janjs8iVraAAIU3YMWtByxZtHKiOGKtSzwFRHDFWpZ+dCcKcQyaSZejFiCcmGBOTR/Jan
js8iVraASwly"
  },
  "id": "6049548642236152503002",
  "orderInformation": {
    "amountDetails": {
      "totalAmount": "3.89",
      "authorizedAmount": "3.89",
      "currency": "GBP"
    }
  },
  "paymentAccountInformation": {
    "card": {
      "type": "001"
    }
  },
  "paymentInformation": {
    "tokenizedCard": {
      "type": "001"
    }
  },
  "processorInformation": {

```



```

    "approvalCode": "3",
    "cardVerification": {
      "resultCodeRaw": "3",
      "resultCode": "2"
    },
    "networkTransactionId": "123456789012345",
    "transactionId": "123456789012345",
    "responseCode": "0",
    "avs": {
      "code": "U",
      "codeRaw": "00"
    }
  },
  "status": "AUTHORIZED",
  "submitTimeUtc": "2020-11-09T20:47:44Z"
}
}
}

```

5.5 STEP 5 - POST to stepUpUrl

If authentication is required (Check Payer Auth Enroll returns status=PENDING_AUTHENTICATION) you must initiate step-up authentication on your front end. This is done by creating an iframe to send a POST request to the step-up URL. The iframe manages customer interaction with the card-issuing bank's Access Control Server (ACS) and 3D Secure version compatibility for 3D Secure 1.0 and 3D Secure 2.x.

Iframe Parameters

- Form POST Action: The URL posted by the iframe is the stepUpUrl field returned by the Check Payer Auth Enrollment call.
- JWT POST Parameter: Use the accessToken field returned by the Check Payer Auth Enrollment call.
- MD POST Parameter: This optional merchant-defined data is returned in the response.

Example Code

The example below shows an example iframe and form.

```

<iframe name="step-up-iframe" height="400" width="400" style="display: none;"></iframe>
<form id="step-up-form" target="step-up-iframe" method="post" action="<<stepUpUrl value>>">
  <input type="hidden" name="JWT" value="<<accessToken value>>" />
  <input type="hidden" name="MD" value="<<optional data returned as is"/>
</form>

```

Add JavaScript to invoke the iframe form POST. Place the JavaScript after the closing </body> tag (see Example below). The JavaScript can be used to invoke the iframe form POST automatically when the Check Payer Auth Enrollment request returns. You may also choose to submit the form at a different time, but you must do it before requesting the Validate Authentication Results call (STEP 8).

```

function showStepUpScreen(stepUpURL, jwt){
  document.getElementById('step_up_iframe').style.display="block";
  document.getElementById('step_up_form').action = stepUpURL;
  document.getElementById('step_up_form_jwt_input').value = jwt;
}

```

```
var stepUpForm = document.getElementById('step_up_form');
if(stepUpForm)submit();
}
```

5.6 STEP 6 - Receive the Authentication Result

After the customer interacts with the issuing bank, you get the session back through the page specified in the **returnURL** field in "Step 4: Check Payer Auth Enrollment". The payload sent to the return URL contains the following data items:

- **TransactionID:** (Same value as the **authenticationTransactionID** field returned by STEP 4 Check Payer Auth Enrollment). This is used in "STEP 7: Validate Authentication Results."
- **MD:** merchant data returned if present in the POST to step-up URL. Otherwise, null.

You must devise a mechanism to return the payload values back to your main page. The following example shows a returnUrl example in PHP and the corresponding function in the main page Javascript:

returnUrl page in iFrame

```
<!DOCTYPE html>
<html>
<head>1
</head>
<body>
<script type="text/javascript">
  var result = <?php echo $_POST['TransactionId']; ?>;
  function closeMe() {
    parent.hideStepUpScreen(result);
  }
  closeMe();
</script>
</body>
</html>
```

Main Page JavaScript

```
function hideStepUpScreen(transactionId) {
  // Hide the iFrame
  ...
  // Call Validate Authentication Results with transactionId
  ...
}
```

5.7 STEP 7 - Validate Authentication Results + Authorization API request

To check whether the cardholder authenticated successfully, you must submit a **Validate Authentication Results API** call **OR** submit a **Process a Payment** request and include a Payer Auth Validation check (**VALIDATE_CONSUMER_AUTHENTICATION**) in the same call. The latter approach is recommended in almost all circumstances, unless there is a specific reason to separate the Payer Authentication flow from the payment flow. With the combined call, if the cardholder DID successfully authenticate, then the authorization request is run and

automatically populated with the correct parameters. If the authentication failed, then the Authorization request will NOT be run and you can ask your customer try again.

5.7.1 Key Request fields

It is recommended to include as much data as possible in your request, as this will increase the chance that the Issuer will not require a challenge screen. See the Developer Center for comprehensive request and response details:

- Process a Payment:
 - https://developer.cybersource.com/api-reference-assets/index.html#payments_payments_process-a-payment
- Check Payer Auth Enrollment:
 - https://developer.cybersource.com/api-reference-assets/index.html?stage=pilot#payer-authentication_payer-authentication_validate-authentication-results

The following request fields are mandatory:

- **consumerAuthenticationInformation** :
 - **authenticationTransactionId**: The **TransactionId** value returned in STEP 6.

The following items are only required for the **Process a Payment** API:

- **processingInformation**:
 - **actionList**: the static value **"VALIDATE_CONSUMER_AUTHENTICATION"**
- **Payment card details**: **FLEX** transient token, **TMS** token or card details.

5.7.2 Request Example 1 - Process a Payment + VALIDATE_CONSUMER_AUTHENTICATION using FLEX token

```
{
  "processingInformation": {
    "actionList": [
      "VALIDATE_CONSUMER_AUTHENTICATION"
    ]
  },
  "orderInformation": {
    "amountDetails": {
      "totalAmount": "3.89",
      "currency": "GBP"
    },
    "billTo": {
      "firstName": "Pauline",
      "lastName": "Evo",
      "address1": "Little House",
      "locality": "Ontheprairie",
      "postalCode": "LH1 OTP",
      "country": "GB",
      "email": "pauline@test.com"
    }
  },
  "tokenInformation": {
    "jti": "1E66U6G8RAEMDQ8GH0JJI9L0L6FIFS2K05LY8T5AI520V4A37PKZP5FAE7B642B88"
  },
  "clientReferenceInformation": {
```

```

    "code": "MOB12345"
  },
  "consumerAuthenticationInformation": {
    "authenticationTransactionId": "DSOxJS51GIRYE4exZhy0"
  }
}

```

N.B The above example assumes that the expiry date fields have been included in the FLEX token.

IMPORT NOTE

The value of “**tokenInformation->jti**” is the value of the JTI claim in the JWT returned by Flex Microform. See **Appendix A - Extracting JTI from Flex Microform Transient Token** for further details.

5.7.3 Request Example 2 - Validate Authentication Results

```

{
  "clientReferenceInformation": {
    "code": "pavalidatecheck"
  },
  "consumerAuthenticationInformation": {
    "authenticationTransactionId": "DSOxJS51GIRYE4exZhy0"
  }
}

```

5.7.4 Key Response fields

The **Process a Payment + VALIDATE_CONSUMER_AUTHENTICATION** API will return the following key fields:

Field	Meaning
status	<p>Indicates status of the transaction. Possible values:</p> <ul style="list-style-type: none"> - AUTHORIZED: - PARTIAL_AUTHORIZED: - AUTHORIZED_PENDING_REVIEW: - AUTHORIZED_RISK_DECLINED: - PENDING_REVIEW: <p>All the above values indicate that the authentication was successful and the authorization request was submitted.</p> <ul style="list-style-type: none"> - DECLINED: <p>Either the authentication failed or the authorization request was declined (e.g. insufficient funds). You can examine <code>consumerAuthenticationInformation:authenticationResult</code> to see whether the authentication was successful.</p> <ul style="list-style-type: none"> - INVALID_REQUEST: <p>An error has occurred.</p>
consumerAuthenticationInformation: authenticationResult	<p>Raw authentication data that comes from the card-issuing bank. Primary authentication field that indicates if authentication was successful and if liability shift occurred. Possible values:</p> <ul style="list-style-type: none"> - 1: Invalid PAREs. - 0: Successful validation. - 1: Cardholder is not participating, but the attempt to authenticate was recorded.

Field	Meaning
	<ul style="list-style-type: none"> - 6: Issuer unable to perform authentication. - 9: Cardholder did not complete authentication.

The **Validate Authentication Results** API will return the following key fields:

Field	Meaning
status	Indicates whether authentication was successful. Possible values: <ul style="list-style-type: none"> - AUTHENTICATION_SUCCESSFUL Authentication was successful. - AUTHENTICATION_FAILED Authentication failed. - INVALID_REQUEST An error has occurred.
consumerAuthenticationInformation: authenticationResult	Raw authentication data that comes from the card-issuing bank. Primary authentication field that indicates if authentication was successful and if liability shift occurred. Possible values: <ul style="list-style-type: none"> -1: Invalid PAREs. 0: Successful validation. 1: Cardholder is not participating, but the attempt to authenticate was recorded. 6: Issuer unable to perform authentication. 9: Cardholder did not complete authentication.

5.7.5 Reply Example 1 - Process a Payment + VALIDATE_CONSUMER_AUTHENTICATION

The following example illustrates the Process a Payment + VALIDATE_CONSUMER_AUTHENTICATION response where the cardholder successfully authenticated and the Authorization was approved:

```
{
  "_links": {
    "void": {
      "method": "POST",
      "href": "/pts/v2/payments/6052697807316824104005/voids"
    },
    "self": {
      "method": "GET",
      "href": "/pts/v2/payments/6052697807316824104005"
    }
  },
  "clientReferenceInformation": {
    "code": "MOB12345"
  },
  "consumerAuthenticationInformation": {
    "indicator": "vbw",
    "eciRaw": "05",
    "cavv": "AAABAWFlmQAAAABjRWWZEEFgFz+=",
    "paresStatus": "Y",
    "authenticationResult": "0",
    "xid": "RFNPeEpTNWxHSVJZRTRleFpoeTA=",
    "cavvAlgorithm": "2",
    "authenticationStatusMsg": "Success",
    "eci": "05",
    "specificationVersion": "1.0.2",
  }
}
```

```

    "token": "Axj//wSTSB40Sy/xCBxFAAIU3YMWTFu3csHCiOGQuQ6gFRHDIXIdWdCcKcFQyaSZeJFiCcmGAaTSB40S
y/xCBxFAtmW6"
  },
  "id": "6052697807316824104005",
  "orderInformation": {
    "amountDetails": {
      "totalAmount": "3.89",
      "authorizedAmount": "3.89",
      "currency": "GBP"
    }
  },
  "paymentAccountInformation": {
    "card": {
      "type": "001"
    }
  },
  "paymentInformation": {
    "tokenizedCard": {
      "type": "001"
    },
    "scheme": "VISA DEBIT",
    "bin": "400000",
    "accountType": "Visa Classic",
    "issuer": "JPMORGAN CHASE BANK",
    "binCountry": "US"
  },
  "processorInformation": {
    "approvalCode": "3",
    "cardVerification": {
      "resultCodeRaw": "3",
      "resultCode": "2"
    },
    "networkTransactionId": "123456789012345",
    "transactionId": "123456789012345",
    "responseCode": "0",
    "avs": {
      "code": "U",
      "codeRaw": "00"
    }
  },
  "status": "AUTHORIZED",
  "submitTimeUtc": "2020-11-13T12:16:21Z"
}

```

5.7.6 Reply Example 2 - Process a Payment + VALIDATE_CONSUMER_AUTHENTICATION

The following example illustrates the Process a Payment + VALIDATE_CONSUMER_AUTHENTICATION response where the cardholder successfully authenticated but the Authorization was declined by the issuer:

```

{
  "_links": {
    "self": {
      "method": "GET",
      "href": "/pts/v2/payments/6052823560456722904001"
    }
  },
  "clientReferenceInformation": {
    "code": "MOB12345"
  },
}

```

```

"consumerAuthenticationInformation": {
  "indicator": "vbv",
  "eciRaw": "05",
  "cavv": "AAABAWFlmQAAAABjRWWZEEFgFz+=",
  "paresStatus": "Y",
  "authenticationResult": "0",
  "xid": "R1hGVFpYdklTazMycEhLT25XVDA=",
  "cavvAlgorithm": "2",
  "authenticationStatusMsg": "Success",
  "eci": "05",
  "specificationVersion": "1.0.2",
  "token": "Axj77wSTSB/NDwByFvPBAAJRHDIfOpACojhkPnUj0iXorAJ8mXpJl6MWIJyYA0mkD+aHgDkLeeCAFDoL
",
},
"errorInformation": {
  "reason": "PROCESSOR_DECLINED",
  "message": "Decline - General decline of the card. No other information provided by the issuing bank."
},
"id": "6052823560456722904001",
"processorInformation": {
  "cardVerification": {
    "resultCodeRaw": "3",
    "resultCode": "2"
  },
  "responseCode": "5",
  "avs": {
    "code": "U",
    "codeRaw": "00"
  }
},
"status": "DECLINED",
"submitTimeUtc": "2020-11-13T15:45:56Z"
}

```

5.7.7 Reply Example 3 - Process a Payment + VALIDATE_CONSUMER_AUTHENTICATION

The following example illustrates the Process a Payment + VALIDATE_CONSUMER_AUTHENTICATION response where the cardholder failed to authenticate, and so the Authorization request was not run:

```

{
  "clientReferenceInformation": {
    "code": "MOB12345"
  },
  "consumerAuthenticationInformation": {
    "eciRaw": "07",
    "paresStatus": "N",
    "authenticationResult": "9",
    "xid": "TkhuekdnZlBKaUtscTNLV2pUVTA=",
    "authenticationStatusMsg": "User failed authentication",
    "specificationVersion": "1.0.2",
    "token": "AxjzbwSTSB/cuYllBK4GAAIBURwyH1PZnQycQE+Xc0ky9GLEE5MA4AAADQw+"
  },
  "errorInformation": {
    "reason": "CONSUMER_AUTHENTICATION_FAILED",
    "message": "Encountered a Payer Authentication problem. Payer could not be authenticated."
  },
  "id": "6052827970086541504006",
  "status": "DECLINED",
}

```

```
"submitTimeUtc": "2020-11-13T15:53:19Z"  
}
```

5.7.8 Reply Example 4 - Validate Authentication Results - Successful Authentication

The following example illustrates the Process a Payment + VALIDATE_CONSUMER_AUTHENTICATION response where the cardholder failed to authenticate, and so the Authorization request was not run:

```
{  
  "clientReferenceInformation": {  
    "code": "pavalidatecheck"  
  },  
  "consumerAuthenticationInformation": {  
    "indicator": "vbv",  
    "eciRaw": "05",  
    "cavv": "AAABAWFlmQAAAABjRWWZEEFgFz+=",  
    "paresStatus": "Y",  
    "authenticationResult": "0",  
    "xid": "dUtORVFvemJWb09yNmVvOVc5eTA=",  
    "cavvAlgorithm": "2",  
    "authenticationStatusMsg": "Success",  
    "eci": "05",  
    "specificationVersion": "1.0.2",  
    "token": "AxijLwSTSCAgLVlInEDCAAJRHDIf/WIAE+hk0ky9GLEE5MAA6wmh"  
  },  
  "id": "6052846956246694904002",  
  "status": "AUTHENTICATION_SUCCESSFUL",  
  "submitTimeUtc": "2020-11-13T16:24:55Z"  
}
```


6 Flex Microform with Payer Authentication

To avoid exposure to card payment details on the merchant server and the additional PCI DSS implications that go with it, it is highly recommended to use the Flex Microform 0.11 to capture the payment details. This frontend integration method returns a token that can be used in place of the card details in all of the APIs used for Payer Authentication. The Microform consists of two hosted fields that can be embedded on the merchant payment page to capture the card number and the CVN. You have full control over the styling of the fields but no access to the card number and CVN entered. The masked PAN and the token are accessible by JavaScript on the page. Full details of how to use the Flex Microform is documented in a separate document: M07: FLEX Microform Integration.pdf.

7 Payer Authentication Testing

To test different Payer Authentication scenarios Smartpay Fuse has a set of test PANs, for Visa, MasterCard and Amex, which trigger different responses from the Payer Authentication platform.

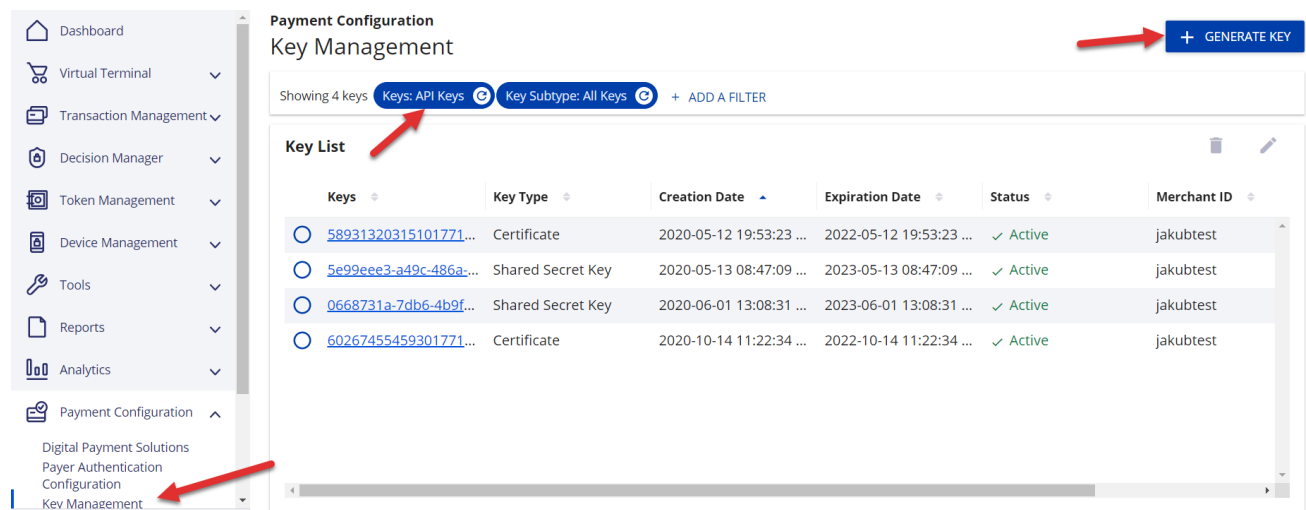
For test PANs please refer to a separate document: "M05 Test Cases.pdf"

As an expiry month for your test transactions, please select always December, for the expiry year – current year + 2. If you are asked to provide a Payer Authentication password, please use "1234".

8 Generating a REST API key

You will need to create a REST Shared Secret security key to be able to make the server-side REST API calls used for Payer Authentication. The following steps show you how this is done:

- 1) Log into EBC, select Payment Configuration -> Key Management, Select **Keys: API Keys** and click on **GENERATE KEY** button.



The screenshot shows the EBC Key Management interface. On the left is a navigation menu with 'Key Management' selected. The main area is titled 'Payment Configuration Key Management' and features a '+ GENERATE KEY' button in the top right. Below this, there are filter buttons for 'Keys: API Keys' and 'Key Subtype: All Keys'. A 'Key List' table displays the following data:

Keys	Key Type	Creation Date	Expiration Date	Status	Merchant ID
58931320315101771...	Certificate	2020-05-12 19:53:23 ...	2022-05-12 19:53:23 ...	Active	jakubtest
5e99eee3-a49c-486a...	Shared Secret Key	2020-05-13 08:47:09 ...	2023-05-13 08:47:09 ...	Active	jakubtest
0668731a-7db6-4b9f...	Shared Secret Key	2020-06-01 13:08:31 ...	2023-06-01 13:08:31 ...	Active	jakubtest
60267455459301771...	Certificate	2020-10-14 11:22:34 ...	2022-10-14 11:22:34 ...	Active	jakubtest

- 2) The **Create Key** window appears. Select **API Cert / Secret** and click **NEXT STEP**.

Key Management
Create Key

1 Select key ————— 2 Select option

Select a key type

What type of key are you creating?

Transaction Processing

API Cert / Secret

NEXT STEP

- 3) In the next menu select **Shared Secret** and then **SUBMIT**.

Key Management
Create Key

1 Select key ————— 2 Select option

Select a key subtype

For API Cert / Secret key type, type of key subtype are you creating?

Certificate
 Shared Secret

Key Configuration

Shared API authenticates using a base-64-encoded transaction key represented in string format. The Shared API generates a key you can copy to your clipboard or download as a text file.

← PREVIOUS STEP CANCEL **SUBMIT**

- 4) In the next window copy or download the Shared secret key, which is the first credential for REST API request, and click on **KEY MANAGEMENT**.

Key Management
Create Key

Shared secret key

You may download this key one time from this screen.

Key
PwnnwFDMM9BnJ/IfPJXZUuDKvMWH7D3rHlwMVaqQmqs=

[↓ DOWNLOAD KEY](#)

KEY MANAGEMENT **CREATE ANOTHER KEY**

- 5) In Key Management window select **Keys: API Keys** option to find newly created key.

Payment Configuration
Key Management + GENERATE KEY

Showing 6 keys Keys: API Keys Key Subtype: All Keys + ADD A FILTER

Key List

Keys	Key Type	Creation Date	Expiration Date	Status
b72b294c-aed1-43fe-978...	Shared Secret Key	2019-01-24 11:22:22 +0000	2022-01-24 11:22:22 +0000	Active
7f6d1f37-ca8c-49d5-afd1...	Shared Secret Key	2019-02-12 19:01:27 +0000	2022-02-12 19:01:27 +0000	Active
0cece072-957e-43e3-925...	Shared Secret Key	2019-02-12 19:01:59 +0000	2022-02-12 19:01:59 +0000	Active
bea7f479-ca78-45fd-bb0d...	Shared Secret Key	2019-02-12 19:02:37 +0000	2022-02-12 19:02:37 +0000	Active
b11a89e3-7eff-4cca-8ff5...	Shared Secret Key	2019-02-14 17:14:32 +0000	2022-02-14 17:14:32 +0000	Active
f1c977c1-760a-4794-862...	Shared Secret Key	2019-02-14 17:26:16 +0000	2022-02-14 17:26:16 +0000	Active

- 6) Click on the link in the **Keys** column to see the key details and copy Key Detail parameter, which is the second credential for REST API request.

Key Management

Key Detail: [f1c977c1-760a-4794-8624-6da9b59521f4](#) ← BACK TO RESULTS

Key Information

Key Type	Activation Date
API: Shared Secret	2019-02-14 17:26:16 +0000
Status	Expiration Date
Active	2022-02-14 17:26:16 +0000
Audit Logs	

You now have the two credentials required for the REST API authentication process:

- Key detail (key id)
- Shared secret key

or invest in any securities to which this presentation relates. Any pricing in this presentation is indicative. Although the statements of fact in this presentation have been obtained from and are based upon sources that Barclays believes to be reliable, Barclays does not guarantee their accuracy or completeness. All opinions and estimates included in this presentation constitute Barclays' judgement as of the date of this presentation and are subject to change without notice. Any modelling or back testing data contained in this presentation is not intended to be a statement as to future performance. Past performance is no guarantee of future returns. No representation is made by Barclays as to the reasonableness of the assumptions made within or the accuracy or completeness of any models contained herein.

Neither Barclays, nor any officer or employee thereof, accepts any liability whatsoever for any direct or consequential losses arising from any use of this presentation or the information contained herein, or out of the use of or reliance on any information or data set out herein.

Barclays and its respective officers, directors, partners and employees, including persons involved in the preparation or issuance of this presentation, may from time to time act as manager, co-manager or underwriter of a public offering or otherwise deal in, hold or act as market-makers or advisers, brokers or commercial and/or investment bankers in relation to any securities or related derivatives which are identical or similar to any securities or derivatives referred to in this presentation.

Copyright in this presentation is owned by Barclays (© Barclays Bank PLC, 2012). No part of this presentation may be reproduced in any manner without the prior written permission of Barclays.

Barclays Bank PLC is a member of the London Stock Exchange.

Barclays is a trading name of Barclays Bank PLC and its subsidiaries. Barclays Bank PLC is registered in England and authorised and regulated by the Financial Services Authority (FSA No. 122702). Registered Number is 1026167 and its registered office 1 Churchill Place, London E14 5HP.

Barclaycard is a trading name of Barclays Bank PLC and Barclaycard International Payments Limited. Barclays Bank PLC is authorised by the Prudential Regulation Authority and regulated by the Financial Conduct Authority and the Prudential Regulation Authority (Financial Services Register number: 122702). Registered in England No. 1026167. Registered Office: 1 Churchill Place, London E14 5HP. Barclaycard International Payments Limited, trading as Barclaycard, is regulated by the Central Bank of Ireland. Registered Number: 316541. Registered Office: One Molesworth Street, Dublin 2, Ireland, D02 RF29. Directors: Paul Adams (British), James Kelly, Mary Lambkin Coyle and Michael Reed (USA). Calls may be recorded for security and other purposes.