3DS1 Integration

3DS 1.0.2 only supported for GPayments SaaS product

3DS 1.0.2 integration is only supported for SaaS clients, and is not available for in-house deployments.

ActiveServer now supports 3DS1 authentication for SaaS clients. To integrate 3DS1 authentication with a merchant's or payment gateway's eCommerce site, the checkout process of the eCommerce site needs to implement a **3DS1 challenge process** which involves an API call to the **ActiveServer** backend, and a page flow to host the challenge process initialised by the ACS.

Summary of ActiveServer's 3DS1 authentication process

The steps below show how to execute a 3DS1 authentication:

1. The **3DS Requestor** (or the merchant checkout process) determines whether a 3DS1 authentication is required.
2. The **3DS Requestor** (or the merchant checkout process) calls **ActiveServer** 3DS1 Auth API with a `ThreeDS1AuthReq` message
3. The **3DS Requestor** redirects current checkout page to the challenge page that is returned by the response `ThreeDS1AuthResp` in step 2
4. The challenge page executes ACS cardholder authentication process and returns the authentication results via a POST form to the result notification URL that was provided by the **3DS Requestor** in message `ThreeDS1AuthReq` in step 2.
5. The authentication result is available for the **3DS Requestor** via the result notification URL. The **3DS Requestor** then processes the result accordingly.

Auth API authentication

Please note the authentication for the 3DS1 Auth API calls uses the same merchant/masterAuth certificate as the existing 3DS2 process, refer to Auth API Authentication for details of certificate usage.

Step 1: Determine 3DS1 authentication

Before executing a 3DS1 authentication process, users may want to check if 3DS1 authentication is necessary or not. The **3DS Requestor** can implement a static protocol routing process that initiates a 3DS1 authentication process based on cardholder or merchant information, or utilise **ActiveServer**'s Enrol API to check if the PAN is enrolled with 3DS2 or not. **3DS Requestor** can proceed with 3DS1 authentication if the Enrol API call returns `00` (Not Enrolled with 3DS2) if 3DS authentication is required.

3DS 1.0.2 authentication determination

Whether a transaction should be authenticated by 3DS1 or 3DS2 is up to the **3DS Requestor** determination. The use of the Enrol API is not mandatory, it is provided as an option for determining whether a 3DS1 authentication is required.

Step 2: Calling 3DS1 Auth API

There are 2 parts of the code for 3DS1 integration: a backend that initialises the Auth API call and hosts a result notification page, and a frontend Javascript method that checks the Auth API call response and redirects/hosts the ACS challenge page.

ActiveServer's 3DS1 integration supports the same languages and frameworks as the 3DS2 integration. Before proceeding with the 3DS1 integration, refer to Integration Introduction to setup your local test environment with the provided demo requestor code and the language/ framework of your choice.

Before calling the 3DS1 Auth API, a proper client certificate is required to setup the mutual authentication TLS communication with **ActiveServer**, refer to Backend Implementation v2 for details.

The backend code snippet below shows how to make a `ThreeDS1AuthReq` call:

Java PHP C# Go

```java
//MainController3DS1.java
  @ResponseBody
  @PostMapping(value = "/3ds1/auth")
  public ThreeDS1AuthResp auth(@RequestBody ThreeDS1AuthReq req) {
    logger.info("3ds1 auth request received: {}", req);
    return threeDS1Service.handleAuthRequest(req);
  }

//ThreeDS1Service.java
  ThreeDS1AuthResp handleAuthRequest(ThreeDS1AuthReq request) {

    //generate the transaction id, this is optional.
    request.setThreeDSRequestorTransID(UUID.randomUUID().toString());

    logger.info("sending 3ds1 auth request to ActiveServer: {}", authUrl);

    HttpEntity<ThreeDS1AuthReq> httpRequest;
    if (config.isGroupAuth()) {
      HttpHeaders headers = new HttpHeaders();
      headers.add(AuthServiceV2.AS_MERCHANT_TOKEN_HEADER,
config.getMerchantToken());
      httpRequest = new HttpEntity<>(request, headers);
    } else {

      httpRequest = new HttpEntity<>(request);
    }

    ResponseEntity<ThreeDS1AuthResp> response = restTemplate
        .postForEntity(authUrl, httpRequest, ThreeDS1AuthResp.class);

    if (response.getStatusCode() == HttpStatus.OK) {

      ThreeDS1AuthResp body = response.getBody();
      logger.info("server returns ok, content: {}", body);

      return body;

    } else {
      ThreeDS1AuthResp body = response.getBody();
      logger.error("server returns error code: {}, content: {}",
response.getStatusCode(),
          body);
      return body;
    }

  }
```

The above code shows the 3DS1 authentication API call is just forwarding the JSON request sent by the frontend to the **ActiveServer** Auth API. The code that loads the mutual authenticated HTTP client and processes the Master Auth Certificate are the same as the 3DS2 code, which is not shown in this guide.

3DS Requestor API authentication

It is up to the **3DS Requestor** implementation to handle its own API authentication with the frontend page flows, and that part is out of scope here.

Prepare the Result Notification URL

When sending `ThreeDS1AuthReq` to **ActiveServer**, the **3DS Requestor** (or the merchant site) must provide a notification page to receive the authentication result from the Issuer ACS. The ACS will post the result (CAVV, ECI etc) to this notification page once the cardholder submits the challenge form.

The **3DS Requestor** can process the result in the backend accordingly. For this guide, we simply show the received authentication result on a result page.

Java PHP C# Go

```
//MainController3DS1.java
  @PostMapping("/3ds1/result")
  public String resultPage(Model model, @RequestBody MultiValueMap<String,
String> body) {
    logger.info("received result: {}", body);

    model.addAttribute("cavv", body.getFirst("cavv"));
    model.addAttribute("cavvAlgo", body.getFirst("cavvAlgo"));
    model.addAttribute("eci", body.getFirst("eci"));
    model.addAttribute("threeDSRequestorTransID",
body.getFirst("threeDSRequestorTransID"));
    return "3ds1/result";
  }
```

As shown in the demo code, the result notification URL is `/3ds1/result`, so the field `callbackUrl` in `ThreeDS1AuthReq` should be set as `https://<the 3ds requestor base URL>/3ds1/result`.

The code snippet below shows how the `callbackUrl` is set on the demo page:

Java PHP C# Go

```java
//MainController3DS1.java
  @GetMapping("/3ds1")
  public String paymentPage(Model model) {
    model.addAttribute("authUrl", config.getAsAuthUrl());
    model.addAttribute("callbackUrl", config.getBaseUrl() + "/3ds1/result");

    logger.info("3ds1 auth page called");
    return "3ds1/auth";
  }
```

Step 3: Process the Auth API response and handle the ACS challenge page

During step 2, **ActiveServer** will process the authentication request with the Directory Server and the ACS internally, and a `ThreeDS1AuthResp` response will be returned by **ActiveServer** if everything goes well. Check the API document overview for the response details.

If `errorCode` in the response is not null, the response is considered as an error, the frontend (or the backend, the actual implementation can be different on how to handle the errors) should handle it as an error and guide the cardholder to retry or recover the authentication process.

If `errorCode` in the response is null, a field `challengeUrl` will be returned. This url is the Issuer ACS' challenge flow page that should be presented to the cardholder who will be asked for a pre-stored credential, or an OTP to verify themselves. The frontend code can redirect the current page to this challenge url or use an iframe to load the challenge url in the browser. Once the challenge page is loaded, and the cardholder submits the challenge form, the ACS will post the result to the `callbackUrl` that was provided in the `ThreeDS1AuthReq` call in Step 2. **3DS Requestor** can process the authentication result and navigate the cardholder to the next stage in the payment process once the `callbackUrl` is triggered.

The Javascript code below shows how to load the challenge url by redirecting the current page, and a simple mechanism to show an error message when the `ThreeDS1AuthReq` returns an error:

Javascript

```
//3ds1/auth.html
<script src="/js/v2/3ds-web-adapter.js"></script>
<script>

  function handleResponse(response) {
    //use the challenge url returned from the response
    if (response.errorCode) {
      console.error("error response", _onError);
      alert("auth request returns error: \n\n" + JSON.stringify(response))
    } else {
      console.log("response:", response)
      window.location.href = response.challengeUrl; //show the challenge url on
current page
    }
  }

  function handleError(response) {
    console.error("error", response);
    alert("auth request returns error: \n\n" + JSON.stringify(response))
  }

  $("#btnAuth").click(function () {

    var authData = objectifyForm($("#authForm").serializeArray());
    console.log(authData);
    doPost("/3ds1/auth", authData, handleResponse, handleError);

  })

  function objectifyForm(formArray) {
    //serialize data function
    var returnArray = {};
    for (var i = 0; i < formArray.length; i++) {
      returnArray[formArray[i]['name']] = formArray[i]['value'];
    }
    return returnArray;
  }

</script>
```

The Javascript snippet shows the challenge url is presented by setting `window.location.href` and the error will be displayed as a simple `alert()` call.

Step 4 & 5: Handle the result notifications

As already mentioned in Step 2, a result notification page must be provided for the `ThreeDS1AuthReq` call. Once the ACS finishes challenge flow, a POST form with authentication results will be posted to the result notification page and this is where the results should be processed by the **3DS Requestor** backend.

Java PHP C# Go

```java
//MainController3DS1.java
  @PostMapping("/3ds1/result")
  public String resultPage(Model model, @RequestBody MultiValueMap<String,
String> body) {
    logger.info("received result: {}", body);

    model.addAttribute("errorCode", body.getFirst("errorCode"));
    model.addAttribute("errorMessage", body.getFirst("errorMessage"));
    model.addAttribute("txStatus", body.getFirst("txStatus"));
    model.addAttribute("cavv", body.getFirst("cavv"));
    model.addAttribute("cavvAlgo", body.getFirst("cavvAlgo"));
    model.addAttribute("eci", body.getFirst("eci"));
    model.addAttribute("threeDSRequestorTransID",
body.getFirst("threeDSRequestorTransID"));
    return "3ds1/result";
  }
```

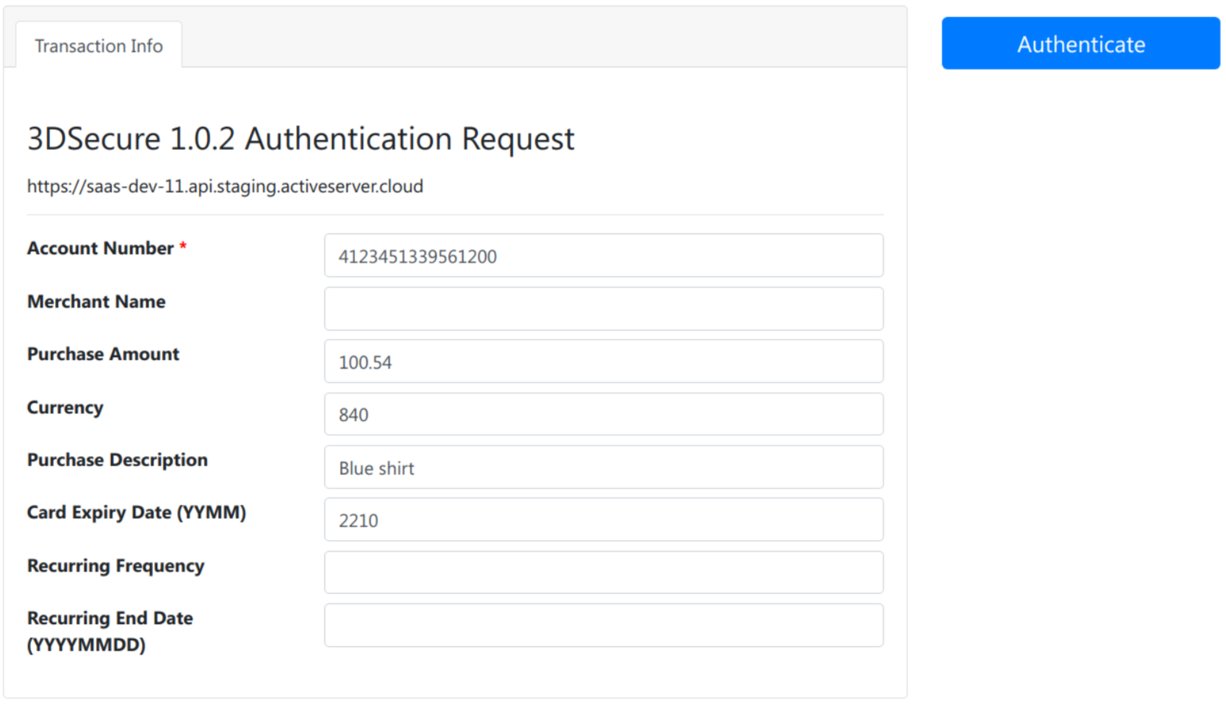
- **cavv**: Cardholder Authentication Verification Value. Required by some Visa regions as proof of 3-D Secure authentication.
- **cavvAlgo**: CAVV algorithm. Required by some Visa regions as proof of 3-D Secure authentication.
- **eci**: Determined by ActiveMerchant. Must be sent to the acquirer through the payment gateway interface.
- **threeDSRequestorTransID**: Transaction id for 3DS1. Used as a “XID” for EnrolReq.
- **txStatus**: the PARes_TX_Status content as defined in the 3DS1 specification.
- **errorCode**: The error code returned. will return “0” if there is no error.
- **errorMessage**: The error message when errorCode is not “0”.

A Walkthrough of 3DS1.0 integration in the demo 3DS Requestor

Now you should be able to run your own 3DS1 requestor with **ActiveServer**, below are screenshots of the 3DS1 authentication process via GPayments 3DS1 TestLab in the demo 3DS Requestor, for your reference.

Submit the authentication request

The demo requestor uses a simple form to populate a `ThreeDS1AuthReq` request and send it as an JSON message to the 3DS Requestor backend. The backend then forwards the request to **ActiveServer**. Please note in the demo code, the auth request form is serialised as JSON data first and then gets posted to the backend.



3DSecure 1.0.2 Authentication Request

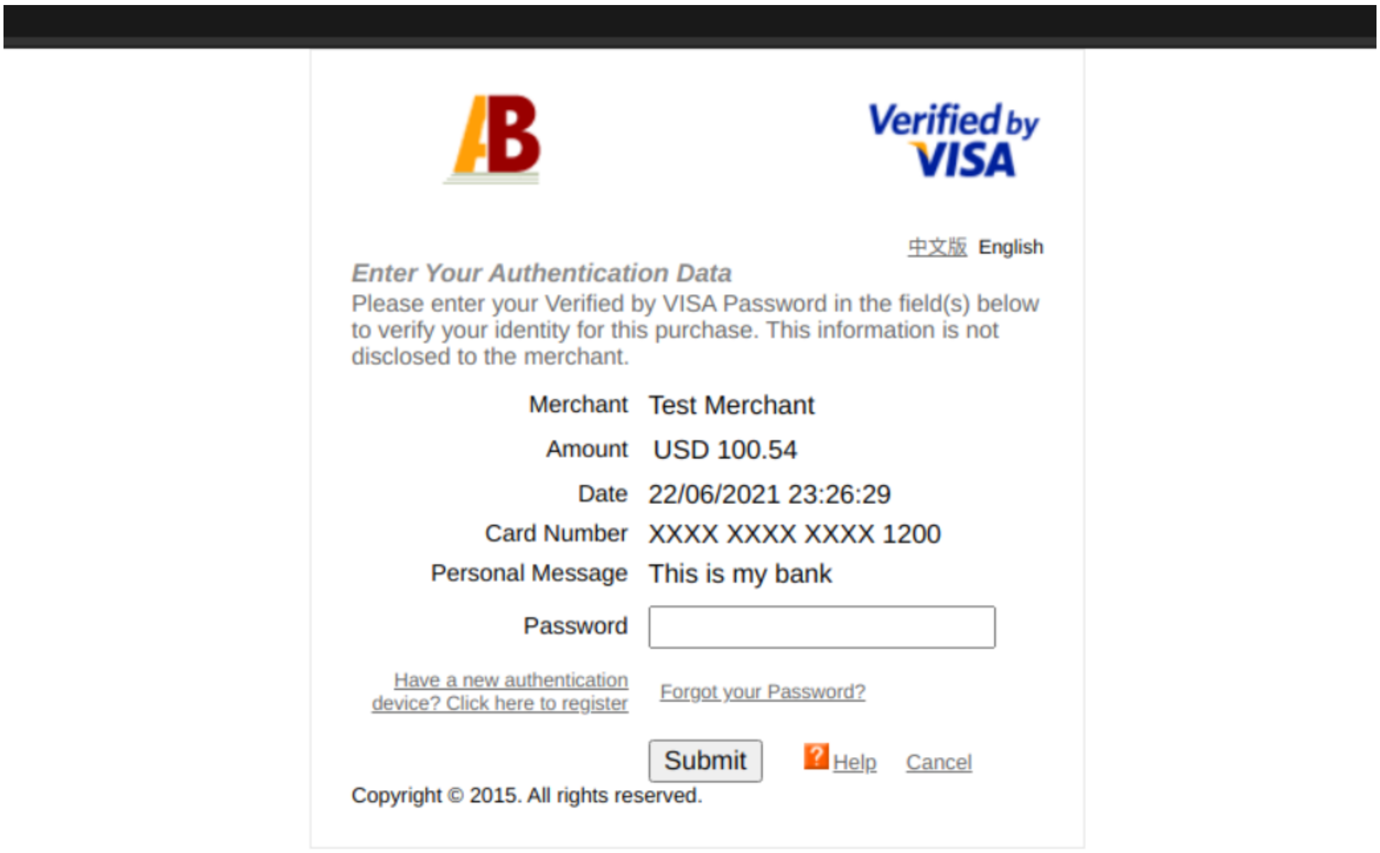
https://saas-dev-11.api.staging.activeserver.cloud

Account Number *	4123451339561200
Merchant Name	
Purchase Amount	100.54
Currency	840
Purchase Description	Blue shirt
Card Expiry Date (YYMM)	2210
Recurring Frequency	
Recurring End Date (YYYYMMDD)	

Challenge page is presented

Once the authentication request is sent to **ActiveServer**, 3DS1 protocol will be processed and the ACS will return a challenge page that should be presented in the browser for the cardholder.

The screenshot below shows a GPayments TestLab ACS challenge page:



中文版 English

Enter Your Authentication Data

Please enter your Verified by VISA Password in the field(s) below to verify your identity for this purchase. This information is not disclosed to the merchant.

Merchant	Test Merchant
Amount	USD 100.54
Date	22/06/2021 23:26:29
Card Number	XXXX XXXX XXXX 1200
Personal Message	This is my bank
Password	

Have a new authentication device? Click here to register

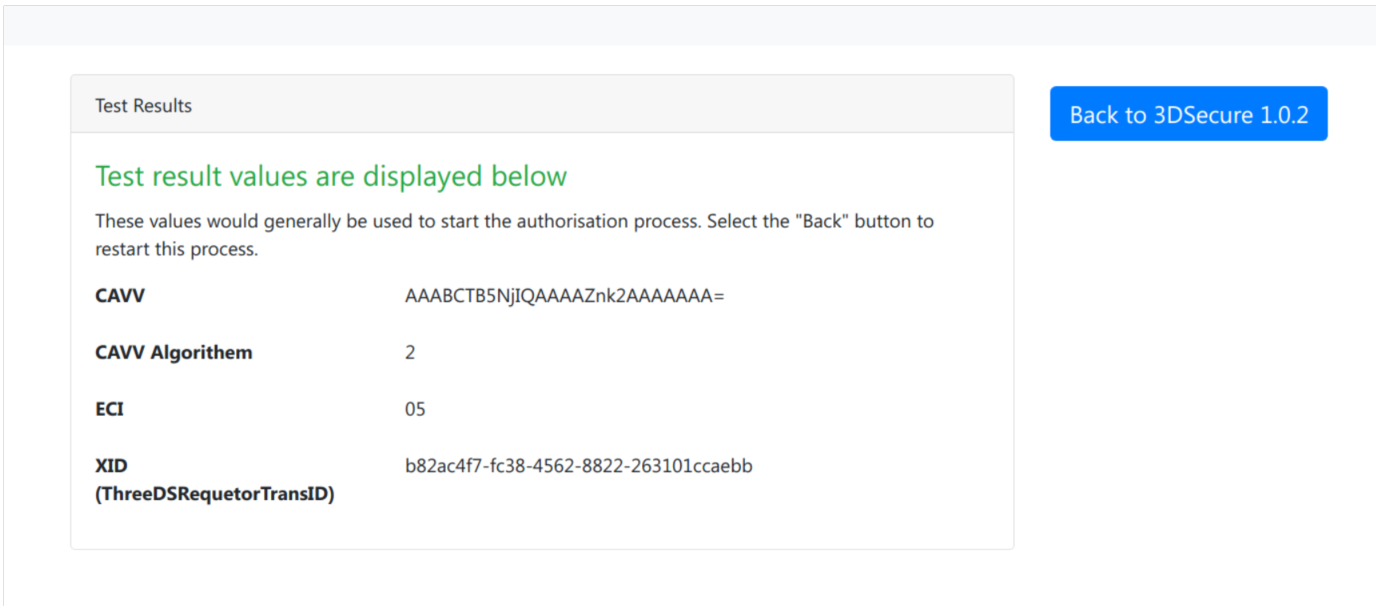
Forgot your Password?

Submit Help Cancel

Copyright © 2015. All rights reserved.

Show the result

Once the cardholder submits their secret on the challenge page, the ACS will verify the input and return the authentication result by posting a form to the `callbackUrl` that was prepared and provided in the initial `ThreeDS1AuthReq` . The demo 3DS Requestor simply shows the result as below:


Test Results

Test result values are displayed below

These values would generally be used to start the authorisation process. Select the "Back" button to restart this process.

CAVV	AAABCTB5NjIQAAAAZnk2AAAAAAA=
CAVV Algorithem	2
ECI	05
XID (ThreeDSRequetorTransID)	b82ac4f7-fc38-4562-8822-263101ccaebb

What's next?

Check out the API Document and learn more about the authentication request/response of 3DS1 in **ActiveServer**