https://verifone.cloud/docs/online-payments/api-integration/server-server-payments-3d-secure-setup-guide/server-server
Updated: 13-Jun-2025

Server-to-Server Payments with 3D Secure

Step 1: (Client-side) Set up your front-end?

Set up your front-end to include the .JS scripts for card encryption and Cardinal Commerce (3DS):

Card encryption: https://cst.jsclient.vficloud.net/verifone.js?

3DS:

- PROD: https://songbird.cardinalcommerce.com/edge/v1/songbird.js
- TEST: https://songbirdstag.cardinalcommerce.com/edge/v1/songbird.js

Throughout the documentation we are using the CST environment. Use the appropriate environment for your account. See more in Getting started.

HTML Example:

```
<head>??
<script src="https://cst.jsclient.vficloud.net/verifone.js"></script>??
<script src=
"https://songbirdstag.cardinalcommerce.com/edge/v1/songbird.js"></script>??

</head>?
```

Step 2: (Server-side) Collect a JWT Token

API Reference: https://verifone.cloud/api-catalog/3d-secure-api#tag/V2/operation/postV2JwtCreate

Make a Post request using your 3D Secure Contract ID to collect a JWT token. This is used later for initializing the 3D Secure script client side.

Request Method: POST?

URL: https://cst.test-gsc.vfims.com/oidc/3ds-service/v2/jwt/create

Body:

```
{
"threeds_contract_id":"{Your 3D-Secure Contract ID}"??
}??
```

Response:

```
{????
??? "threeds_contract_id": " {Your 3D-Secure Contract ID}",????
??? "jwt":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJqdGkiOiIzNjUzMTU0Yy1jNzczLTRhNjAtYWNkMi
???
}?
```

The JWT will expire after 2 hours and it is a single time use only.

?Pass the JWT token to your front-end for later use.

Step 3: (Client-Side) Initiate your Card Form?

Once you have a JWT token, you should generate your card collection form/page to begin the 3D Secure assessment and allow the customer to enter their credit card details at the same time.

Step 4: (Client-side) Configure the 3D Secure Script

Run the Cardinal.configure() function with your desired configuration options. For example:

```
Cardinal.configure({
    timeout: 6000,
    maxRequestRetries: 3,
    logging:{
        level: 'verbose'
    }
});
```

- Root Level Object
- Logging Object
- Button Object
- Payment Object

Field	Type	Default	Description
timeout	int	8000	The time in milliseconds to wait before a request to Centinel API is considered a timeout.

Field	Type	Default	Description
extendedTimeout	int		extendedTimeout is only used in the event of the first request timing out. This configuration allows the merchant to set the timeout (in milliseconds) for subsequent retry attempts. (This configuration would be useful when the merchant wants to set higher timeout values on requests). If the value for the extendedTimeout is set to less than 4000 milliseconds, then the value will be automatically reset to 4000 milliseconds.
maxRequestRetries	int	1	How many times a request should be retried before giving up as a failure.

Field	Type	Default	Description
level	string	off	The level of logging to the browser console. Enable this feature to help debug and implement Songbird. **Possible Values**: • **off** - No logging to console enabled. This is the setting to use for production systems. • **on** - Similar to info level logging, this value will provide some information about whats occurring during a transaction. This is recommended setting for merchants implementing Songbird. • **verbose** - All logs are output to console. This method can be thought of as debug level logging and will be very loud when implementing Songbird, but is the level needed when getting support from the Cardinal team.

Field	Type	Default	Description
containerId	string	Cardinal-Payments	The HTML Id value of the container to inject all payment buttons into.

Field	Type	Default	Description
view	string	modal	What type of UI experience to use when Songbird injects payment brand UI elements into the page. **Possible Values**: • **modal** - Render as a modal window. This view type renders the payment brand over your page, making it feel separate from your page.

Field	Type	Default	Description
framework	string	Cardinal	What kind of view framework should be used to render the payment brand. If your site is using a supported framework and you have custom styles applied to it, we will use that framework to make keep the consistent look and feel of your site. When using any other frameworks than 'cardinal' your site is responsible for including the framework assets including CSS, JavaScript, and any other additional files needed. **Possible Values**: • **cardinal** - Use the custom Cardinal view framework built and maintained by CardinalCommerce. Songbird will handle all UI rendering and styles, no additional work is needed. • **inline** - Render inline to the page. This view type embeds the payment brand into the page making it feel like it's a part of your website. View the guide for implementation here: Inline Display Method for 3D Secure • **bootstrap3** - Use bootstrap 3 modal to render the UI elements. Please note that you are responsible for importing any necessary files for that framework.
displayLoading	boolean	false	A flag to enable / disable a loading screen while requests are being made to 3DS Server API services. This can provide feedback to the end user that processing is taking place and they should not try to reload the page, or navigate away. **Possible Values**: • **false** - Disables the loading screen • **true** – Enables the loading screen
displayExitButton	boolean	false	Will display an X icon in the corner of the modal window to allow for end users to close the authentication modal without completing it. Clicking the close button will result in the payments.validated event to be triggered with a “10011 error, Canceled by user". **Possible Values**: • **false** - Disables the exit icon on the modal • **true** - Enables the exit icon on the modal

Step 5: (Client-side) Setup Event Listeners

payments.setupComplete()?

This listener will run after the payment setup has successfully been completed. SetupCompleteData will contain a session ID to be used for the 3DS lookup.?

```
Cardinal.on('payments.setupComplete', function(setupCompleteData){??
// pass setupCompleteData.sessionId server side to make the lookup API call

});?
```

payments.validated()

Payments Validated allows you to capture the different outcomes of the flow and handle them accordingly.?

```
Cardinal.on("payments.validated", function (data, jwt) {??
??? switch(data.ActionCode){??
????? case "SUCCESS":??
????? // Handle successful transaction, send JWT to backend to verify??
????? break;??

????? case "NOACTION":??
????? // Handle no actionable outcome??
????? break;??
????? case "FAILURE":??
????? // Handle failed transaction attempt??
????? break;??
???????
????? case "ERROR":??
????? // Handle service level error??
????? break;??
? }??
});?
```

Response Data and Outcome definitions

Type	Description
ActionCode	The resulting state of the transaction. Possible values: • SUCCESS - The authentication was successful. Details from the “Payment” Object can be used to perform a payment. • NOACTION - The API calls were completed and there is no further actionable items to complete. This can indicate that the card holder is not enrolled in 3D Secure or it could indicate a validation error was encountered. It is recommended that the rest of the response is reviewed to determine what has occurred. • FAILURE - The authentication resulted in an error. For example, this would indicate that the user failed authentication or an error was encountered while processing the transaction. • ERROR - A service level error was encountered. These are generally reserved for connectivity or API authentication issues. For example, if your JWT was from the wrong environment, or 3DS services are unavailable.
Validated	This value represents whether the authentication was successful or not.
ErrorNumber	Application error number. A non-zero value represents the error encountered while attempting the process the message request.

Type	Description
ErrorDescription	Application error description for the associated error number.
Payment	Payment Object, see below for details.

In the case of a SUCCESS outcome, the details returned from data.Payment.ExtendedData are used for payment in Step 12.

Example for SUCCESS outcome:


```
{
    "Type": "CCA",
    "ExtendedData": {
        "Amount": "1000",
        "AuthorizationPayload":
"eyJjb250YWluZXJWZXJzaW9uIjoiMSIsImVjaSI6IjA1IiwiYXV0aGVudGljYXRpb25WYWx1ZSI6IkF
,
        "CAVV": "AAIBYYNoEwAAACcKhAJKdQAAAAA=",
        "CurrencyCode": "554",
        "ECIFlag": "05",
        "ThreeDSVersion": "2.2.0",
        "PAResStatus": "Y",
        "SignatureVerification": "Y"
    },
    "ProcessorTransactionId": "LUj6k3aJ51K6pUh2UeV1"
}
```


Step 6: (Client-side) Initialize the 3D Secure Script?

Initialize the 3D Secure Script using the JWT Token generated from Step 2.

This will begin the 3D Secure process:


```
Cardinal.setup('init', { jwt:??
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJqdGkiOiIzNjUzMTU0Yy1jNzczLTRhNjAtYWNkMi0
??
});?
```


Step 7: (Client-side) Card Number detection

While the customer is entering their card number, use one of the following “BIN detection” methods to positively impact authentication success rates, increase the opportunity for frictionless authentication, and reduce customer abandonment.

Option 1 (Recommended): Field Decorator

This implementation is the simplest and recommended approach and requires the least amount of work to complete. Simply add a new attribute to the input field to identify which field it maps to within the Order Object. The credit card number is mapped to the AccountNumber field, so for bin detection we would pass AccountNumber to the attibute data-cardinal-field.

Example:

```
<input type="text" data-cardinal-field="AccountNumber" id=
"creditCardNumber" name="creditCardNumber" />
```

Option 2: Event Based

The bin.process event is the recommended BIN Detection implementation. It provides you, or integrator, the greatest flexibility to initiate device profiling wherever they prefer in their purchase flow. It is best practice to initiate bin.process immediately upon receiving the customer's card number. Whenever possible, provide a minimum of the first 9 BIN digits of the customer's card number on bin.process. Merchants that provide fewer than the first 9 digits are at risk of running the incorrect issuer Method URL.

Example:

```
Cardinal.trigger("bin.process", '1234567894561237')
  .then(function(results){
    if(results.Status) {

// Bin profiling was successful. Some merchants may want to only move forward wi

    } else {
        // Bin profiling failed
    }

// Bin profiling, if this is the card the end user is paying with you may start

    Cardinal.start('cca', myOrderObject);
})
  .catch(function(error){
    // An error occurred during profiling
})
```

You may have to trigger the bin events more than once if the end user is able to change their card number at the point where Songbird is integrated. Songbird will only profile any given bin a single time, once profiling is completed Songbird will return a success status. It is important that BIN Detection is completed on the final card used for the purchase.

The event will resolve when bin profiling was successful or failed with a JSON object describing the outcome.

Step 8: (Client-side) Collect the sessionId

The sessionId is a unique ID which represents the device profile of the web user. This is used as part of the 3D Secure Authentication process.

If the setup was successful, the event listener in Step 4, payments.setupComplete(), will be triggered, and return the session ID.?

Within the response of setupCompleteData, there will be a value called sessionId:

setupCompleteData.sessionId

Returns:

0_6e0fedb-d642-47d1-88e2-b12a59ffe39e?

Step 9: (Client-side) Collect Card Details with Verifone.JS?

Use the Verifone.js script to securely encrypt the card details before transmitting it to Verifone.

Collect your Secure card capture key, and set this to a variable, for example:?

?const encryptionKey = '{Secure Card Capture Key}';??

Capture the cardholder details from the front-end and set this up as an object:

```
const card = {?
??? "cardNumber": form.cardNumber.value,?
??? "expiryMonth": form.expiryMonth.value,?
??? "expiryYear": form.expiryYear.value,?
??? "cvv": form.cvv.value,?
? };
```

?Then call the verifone encryption method, passing in both the cardDetails and encryptionKey as parameters.?

Calling the method:

```
verifone.getEncryptedCardDetails(card, encryptionKey)?
```

This method returns a Promise containing the encryptedCard field.

```
verifone.getEncryptedCardDetails(card, encryptionKey).then(data => console
.log(data.encryptedCard));?
```

Response:

```
Ls0LS1CRUdJTiBQR1AgTUVTU0FHRS0tLS0tDQpWZXJzaW9uOiBPcGVuUEdQLmpzIHY0LjEwLjkNCkNvbW
???
```

These are the card details encrypted?. Pass this to your back end for processing payments.

Step 10: (Server-side) Perform a 3DS Lookup?

API Reference: https://verifone.cloud/api-catalog/3d-secure-api#tag/V2/operation/postV2Lookup

Make a POST request to Verifone using the encrypted card, device info (sessionId) and other data points to receive the 3D Secure Response.

The device_info_id is the same value as the sessionId returned from step 8.

Request Method: POST

URL: https://cst.test-gsc.vfims.com/oidc/3ds-service/v2/lookup

Body:

```
{??
??? "amount": 100,??
??? "billing_first_name": "first_name",??
??? "billing_last_name": "last_name",??
??? "billing_address_1": address line 1",??
??? "billing_city": "City",??
??? "billing_country_code": "AU",??
??? "encrypted_card": "{{EncryptedCardValue==}}",??
??? "public_key_alias": "{{public_key}}",??
??? "currency_code": "{{currency}}",??
??? "device_info_id": "0_6e00fedb-d944-47d1-89e2-b12a59ffe39x",??
??? "email": "email@verifone.com",??
??? "merchant_reference": "Order number 1234",??
??? "threeds_contract_id": "{{3DS Contract ID}}"??
}
```

Response:

```
?{??
??? "acs_transaction_id": "9d08e1ba-0240-4283-813a-a3772d80de0e",??
??? "acs_url": "{ACS_URL},??
??? "authentication_id": "c923575f-86e4-45cf-9a43-d1ede3da3ac1",??
??? "challenge_required": "N",??
??? "card_brand": "Visa",??
??? "ds_transaction_id": "e3b59e11-5863-4c4a-aa34-cc13bab4f320",??
??? "eci_flag": "07",??
??? "enrolled": "Y",??
??? "error_no": "0",??
??? "order_id": "8001840452769160",??
??? "pares_status": "C",??
??? "payload": "
eyJtZXNzYWdlVHlwZSI6IkNSZXEiLCJtZXNzYWdlVmVyc2lvbiI6IjIuMi4wIiwidGhyZWVEU1NlcnZl
",??
??? "signature_verification": "Y",??
??? "threeds_version": "2.2.0",??
??? "transaction_id": "mm1WQNPXlhUnAYGyjNY1"??
}?
```

Step 11A: (Server-side) Successful 3D Secure Lookup

If the lookup attempt was successful, and the details provided match the card holder details on record, the field **“pares_status” will be “Y”**.

You can proceed directly to Step 12 to perform a payment using the details from the lookup request.

Step 11B: (Client-side) Pares_status = “C” Continue to the authentication step?

In some cases, a one-time pin or “Step-up” challenge is required to authenticate the customer.

Cardinal.continue will only work after the payments.setupComplete event has been triggered.

Cardinal.continue is suggested to be run later in the flow if payments.setupComplete is not triggered yet.

Example:

```
Cardinal.continue('cca',??
{??
?"AcsUrl": “{ACS_URL}”,??
??? "Payload":
"eyJtZXNzYWdlVHlwZSI6IkNSZXEiLCJtZXNzYWdlVmVyc2lvbiI6IjIuMi4wIiwidGhyZWVEU1NlcnZ
??
?},??
?{??
??? "OrderDetails":{??
???????? "TransactionId" : "mm1WQNPXlhUnAYGyjNY1"??
?????????????????????? }??
??????? }??
);?
```

This will present the 3D Secure modal window to the customer:
3D Secure modal window

For Sandbox testing, the code will always be 1234 and not sent to a phone number.

After completion of the one-time pin challenge, the 3D Secure plugin will return an outcome from one of the **SUCCESS** Event Listener setup in Step 5, for example:

```
{
    "Type": "CCA",
    "ExtendedData": {
        "Amount": "1000",
        "AuthorizationPayload":
"eyJjb250YWluZXJWZXJzaW9uIjoiMSIsImVjaSI6IjA1IiwiYXV0aGVudGljYXRpb25WYWx1ZSI6IkFI
",
        "CAVV": "AAIBYYNoEwAAACcKhAJKdQAAAAA=",
        "CurrencyCode": "554",
        "ECIFlag": "05",
        "ThreeDSVersion": "2.2.0",
        "PAResStatus": "Y",
        "SignatureVerification": "Y"
    },
    "ProcessorTransactionId": "LUj6k3aJ51K6pUh2UeV1"
}
```

Understanding the Impact of the different 3D Secure Responses

If **"PAResStatus"** is one of the following:
“Y" – 3DS is Successful
“A” – 3DS was Attempted

And **“SignatureVerification”** is:
“Y” - Contents of the message can be trusted

And **“enrolled”** is:
“Y” - Cardholders bank is participating in 3D Secure Process.

This will result in a liability shift off the merchant.

If **"PAResStatus”** is one of the following:
“N” - Failed
“U” - Unavailable
“R” - Rejected
“C” - Challenge required (Temporary status)

Or a blank value

Or **“SignatureVerification”** is:
“N” - Signature verification is invalid

OR **“enrolled”** is:
“N” - Cardholders bank is not participating in 3D Secure
“U” - Enrollments status unavailable
“B” - Enrollment status was bypassed

The liability of the transaction will remain with the merchant.

This is not a complete guarantee of liability shift. The information presented in this document is based on published Card Associations (Visa, Mastercard, AMEX, ProtectBuy, and JCB) Operating Rules and Regulations, and may be subject to change.

Step 12: (Server-side) Perform a Server-To-Server Payment with 3D Secure?

Using the data from the lookup and if applicable, the step-up challenge response, you may perform a 3D Secure authenticated payment. See the example API request below:

API Reference: https://verifone.cloud/api-catalog/verifone-ecommerce-api#tag/Ecom-Payments

Request method: POST

URL: https://cst.test-gsc.vfims.com/oidc/api/v2/transactions/card?

Example API Request:

```
{??
? ? "currency_code": "{{currency}}",??
? ? "amount": 1000,??
? ? "merchant_reference": "VF Test",??
? ? "payment_provider_contract": "{{ppc}}",??
? ? "card_brand": "VISA",??
```

```
"LS0tLS1CRUdJTiBQR1AgTUVTU0FHRS0tLS0tDQpWZXJzaW9uOiBPcGVuUEdQLmpzIHY0LjEwLjkNCkN
,??
? ? "threed_authentication": {??
? ? ? ? "eci_flag": "05",??
? ? ? ? "enrolled": "Y",??
? ? ? ? "cavv": "AAIBBYNoEwAAACcKhAJkdQAAAAA=",??
? ? ? ? "pares_status": "Y",??
? ? ? ? "threeds_version": "2.2.0",??
? ? ? ? "ds_transaction_id": "4eaa10ef-e5e4-4b4c-8aef-29439e450b60",??
? ? ? ? "signature_verification": "Y",?
? ? ? ? "error_desc": "Success",??
? ? ? ? "error_no": "0"??
? ? }
}??
```


Response:


```
{?
? ? "id": "3e5baa3a-cdcd-4b0a-b23d-9b4c0528ca62",?
? ? "payment_provider_contract": "d1f0f6ab-1d40-44ae-b16b-8f09fe6fd77f",?
? ? "amount": 1000,?
? ? "blocked": false,?
? ? "merchant_reference": "VF Test",?
? ? "payment_product": "CARD",?
? ? "status": "AUTHORIZED",?
? ? "arn": "SIMULATORBCLDCG4DGS5MSDEV46JV",?
? ? "scheme_reference": "170320243e5baa3acdcd4b0ab23d9b4c0528ca62",?
? ? "created_by": "7b360b69-1787-40dd-ac56-fb3b8b93f230",?
? ? "cvv_present": true,?
? ? "cvv_result": "4",?
? ? "stored_credential": {?},?
? ? "details": {?
? ? ? ? "auto_capture": true?
? ? },?
? ? "reason_code": "00",?
? ? "shopper_interaction": "ECOMMERCE",?
? ? "stan": "157449",?
? ? "threed_authentication": {?
? ? ? ? "eci_flag": "05",?
? ? ? ? "enrolled": "Y",?
? ? ? ? "cavv": "AAIBBYNoEwAAACcKhAJkdQAAAAA=",?
? ? ? ? "pares_status": "Y",?
? ? ? ? "threeds_version": "2.2.0",?
? ? ? ? "ds_transaction_id": "4eaa10ef-e5e4-4b4c-8aef-29439e450b60"?
? ? },?
? ? "reversal_status": "NONE",?
? ? "additional_data": {?
? ? ? ? "initiator_trace_id": "157449"?
? ? },?
? ? ? ? "card_details": {?lia
? ? ? ? "masked_card_number": "411111****1111",?
? ? ? ? "expiry_year": 2030,?
? ? ? ? "expiry_month": 12?
? ? },?
? ? "balance_amount": 0?
}?
```

Your Server-to-server, 3D Secure Authenticated transaction is now complete.

Additional steps

Refund a transaction

Depending on your supported acquirer and according to the settlement time. A payment can be refunded fully or partially via Verifone Central, or via the Ecommerce API using the refund payment API call.

Notification methods

Set up notifications in Verifone Central to receive transaction events via email or webhook URL's. Leverage notifications to receive transaction results to different systems at the time of payment.

Advanced Payment flow with Preauthorization

To perform a preauthorization, two fields need to be specified in the payment request in Step 12:

- capture_now : false
- auth_type : "PRE_AUTH"

Once the Preauth is authorized, it can be captured for settlement, or cancelled (voided) using payment actions through Verifone Central or the eCommerce API.

Adding Tokenization

Set up a token scope in Verifone Central to set up a "Vault" with Verifone to store tokens. Request a token by adding the "Token_preference" object to your API request.

Adding Stored credentials

After setting up Tokenization, use the Stored Credential Framework to send MIT or CIT-approved transactions.