ParadoxLabs CyberSource Payments: User Manual

Version 2.0 – For Magento® 2.3+ – Updated 2025-01-10

Table of Contents

Installation

We strongly recommend installing, configuring, and testing all extensions on a development website before installing and using them in production.

Installing an extension requires familiarity with your server's command line.

Please use the Composer package manager to install and update this extension.

In SSH, from your site root, run the following commands:

```
composer require paradoxlabs/cybersource:*
php bin/magento module:enable -c ParadoxLabs_TokenBase ParadoxLabs_CyberSource
php bin/magento setup:upgrade
```

If your site is in production mode, you will need to run these commands next to recompile sources:

```
php bin/magento setup:di:compile
php bin/magento setup:static-content:deploy
```

Once complete, the extension should be available. See the Configuration section below to continue.

Option: Install via download package

If you obtained an extension download package (from our store or Github), instead of the composer require line:

Upload all files within the **upload** folder into the root directory of Magento.

Folder in Download		Folder on Server
/upload/app/	→	/app/

Then continue with the commands to enable the module, run setup, and recompile Magento.

Updating the Extension

All extension updates are free. Just follow these directions to update to the latest version.

If you installed with composer, update with the following commands, in SSH at your site root:

```
composer update paradoxlabs/*
php bin/magento setup:upgrade
```

This will download and update to the latest extension version compatible with your system.

If your site is in production mode, run these additional commands to recompile:

```
php bin/magento setup:di:compile
php bin/magento setup:static-content:deploy
```

Option: Update via download package

Upload all files within the **upload** folder into the root directory of Magento.

Folder in Download		Folder on Server
/upload/app/	→	/app/

Then run setup and recompile, per the commands above.

Connecting A New CyberSource Account

Before proceeding: Contact CyberSource to sign up for merchant account if you don't have one already. You will need to go through the account setup and activation process before you can accept real payments.

Configuring the CyberSource payment method is a bit of a process: There are four separate APIs involved that you need to generate or share the API credentials for. We're going to walk you through the whole process though. Click the right options and it'll be done in no time. We'll go through each section one at a time.

Note, the configuration steps should be the same for a sandbox account or production, but sandbox accounts and production accounts are entirely separate. If you want to test, you must have a dedicated sandbox account. You can create one at: https://ebc2.cybersource.com/ebc2/registration/external

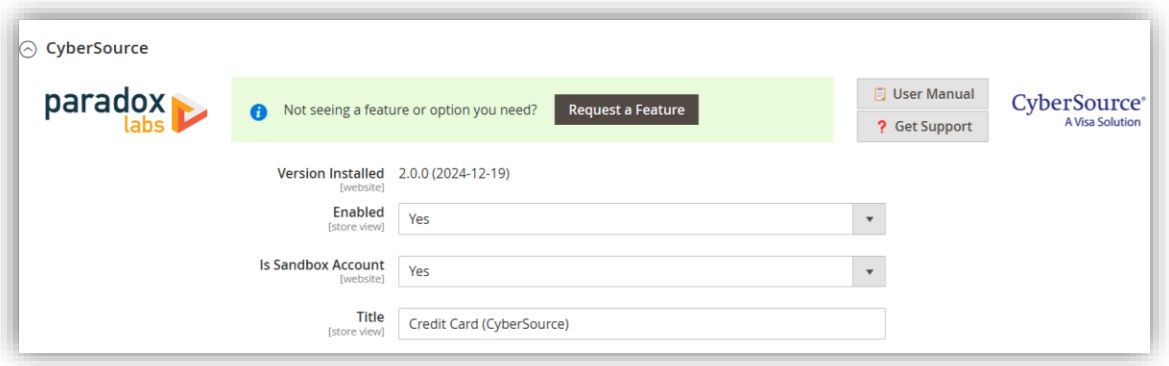
Also note, many CyberSource features require CyberSource configuration and enablement. If you don't see something, please contact CyberSource.

This integration requires payment processing AND Token Management Service to be activated. It can also make use of Decision Manager (or Fraud Management Essentials), Payer Authentication, and Account Updater if they are enabled on your account.

Without further ado:

Step 1. Finding the Magento configuration

Open your Admin Panel and go to **Admin > Stores > Settings > Configuration > Sales > Payment Methods**. Toward the bottom of the page, you'll find a 'CyberSource' settings section like this:

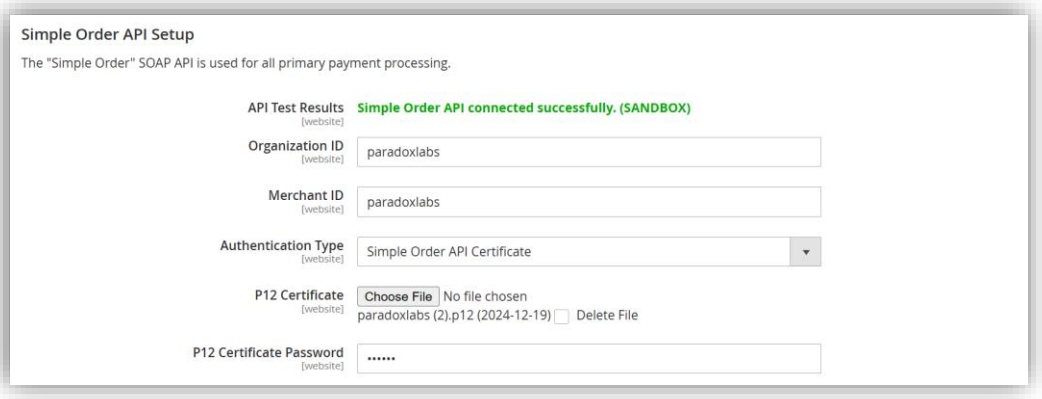


When you see this, you're at the right place. This is where you'll enter all of the API credentials, and set additional payment method configuration options. For the top section:

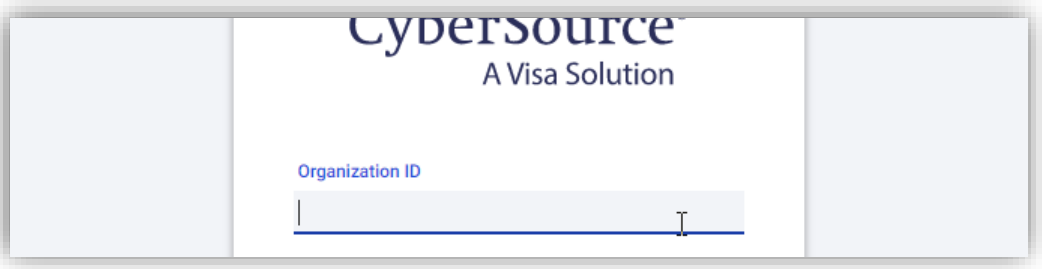
1. Leave 'Enabled' set to No until we're done.
2. Set 'Is Sandbox Account' to Yes if you're setting up a test account, or No if this is for live payment processing.
3. If you'd like, change the Title while you're here.

Let's move on to API setup.

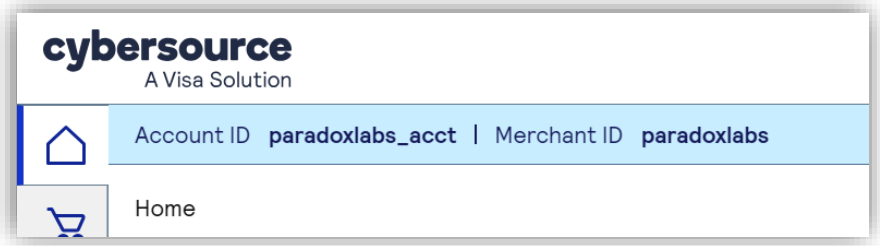
Step 2. Simple Order API Setup



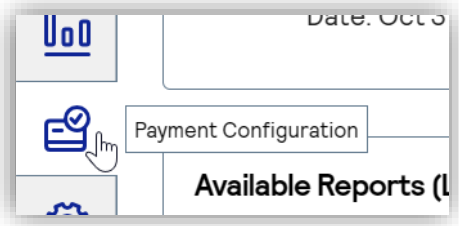
Go to your **CyberSource Enterprise Business Center (EBC) login form**. Before you log in, take note of your **Organization ID**: Enter the same value you use to log in as your Organization ID in Magento.

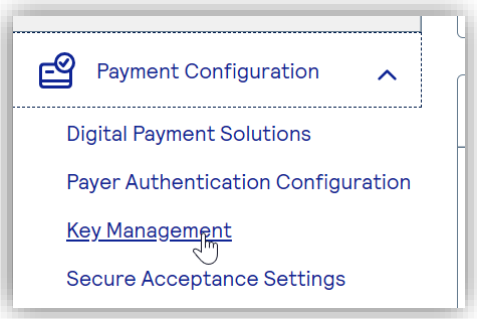


Now log in to EBC. At the top, you'll see a **Merchant ID**. Enter that same value as your Merchant ID in Magento.



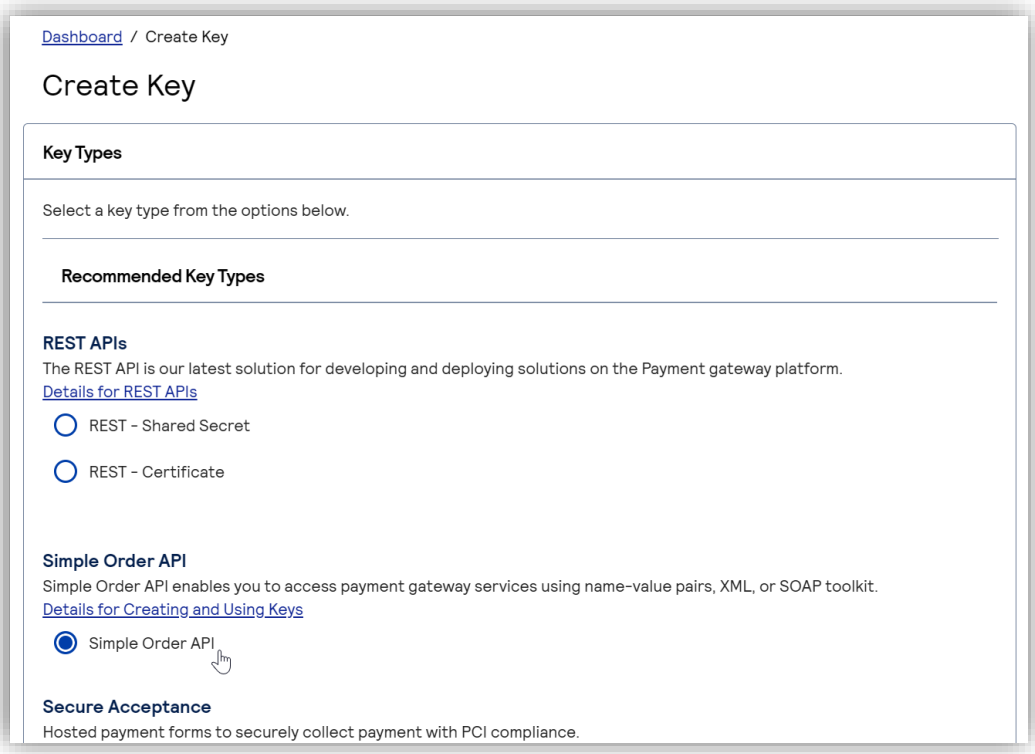
Finally, in EBC, click **Payment Configuration** (second to last menu icon), then go to **Key Management**:



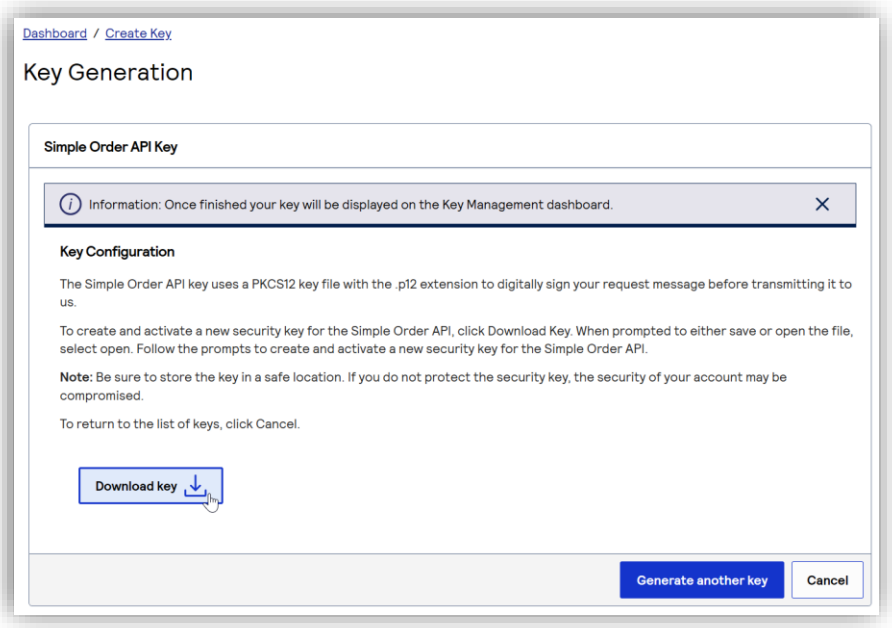


Once you've loaded the Key Management page, click **Generate Key** at the top right:

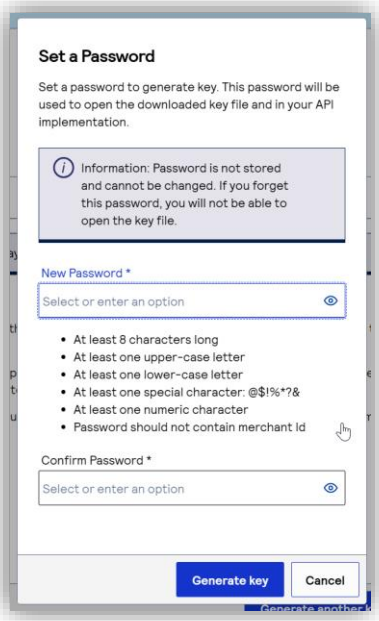
You'll be taken to an API Key creation form. Select **Simple Order API**, and then click 'Generate Key' to continue:



On the next page, click 'Download key':



To continue, you must create a password for the certificate. You don't need to remember it, but you do need to enter the same password into the Magento settings. We suggest randomly generating a secure password.



Enter a password, and put the same password in Magento as **P12 Certificate Password**. Then click 'Generate key'.

The API key will be automatically downloaded to your computer as a *.p12 file. Upload this to the Magento settings page into the **P21 Certificate** setting.

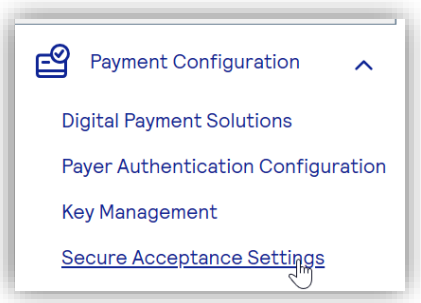
Now save your Magento configuration. If you've done everything correctly, you should see a green message: **Simple Order API connected successfully.** If you get a red error message instead, fix any problems it mentions, and recheck the values you entered.

One down, three to go.

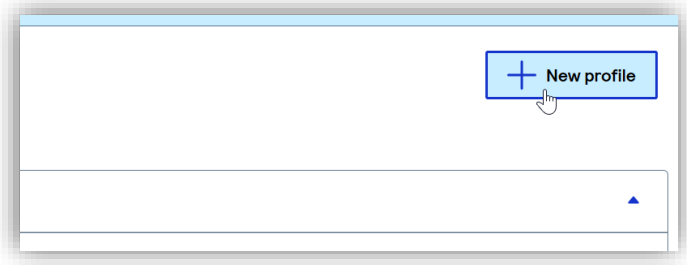
Step 3. Secure Acceptance Checkout Setup

This step is the most involved: You need to create a Secure Acceptance Checkout profile for the checkout credit card form. This determines the credit card types you accept, fields that are displayed and required, form colors, and more.

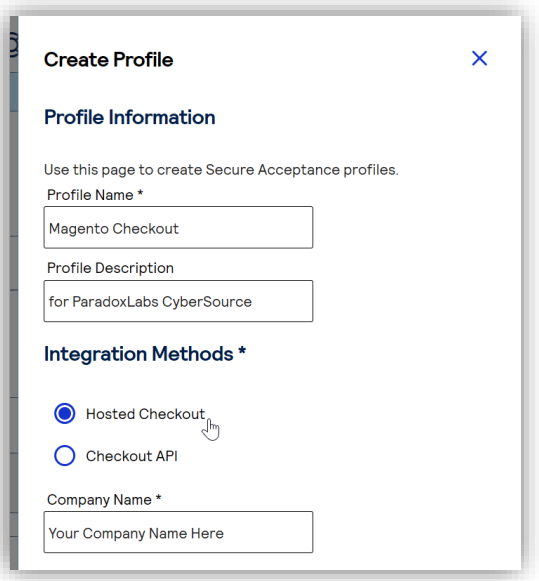
Go back to **CyberSource EBC**, and click **Secure Acceptance Settings** on the sidebar menu:



When the Secure Acceptance Settings page has loaded, click **New Profile** at the top right:

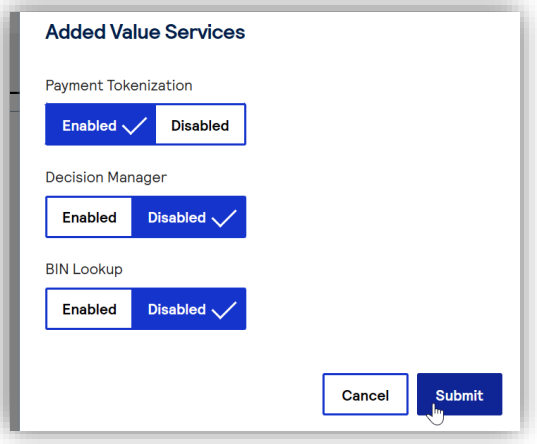


A form will pop up. At the top, enter a profile name and description, select **Hosted Checkout**, and enter your company name.



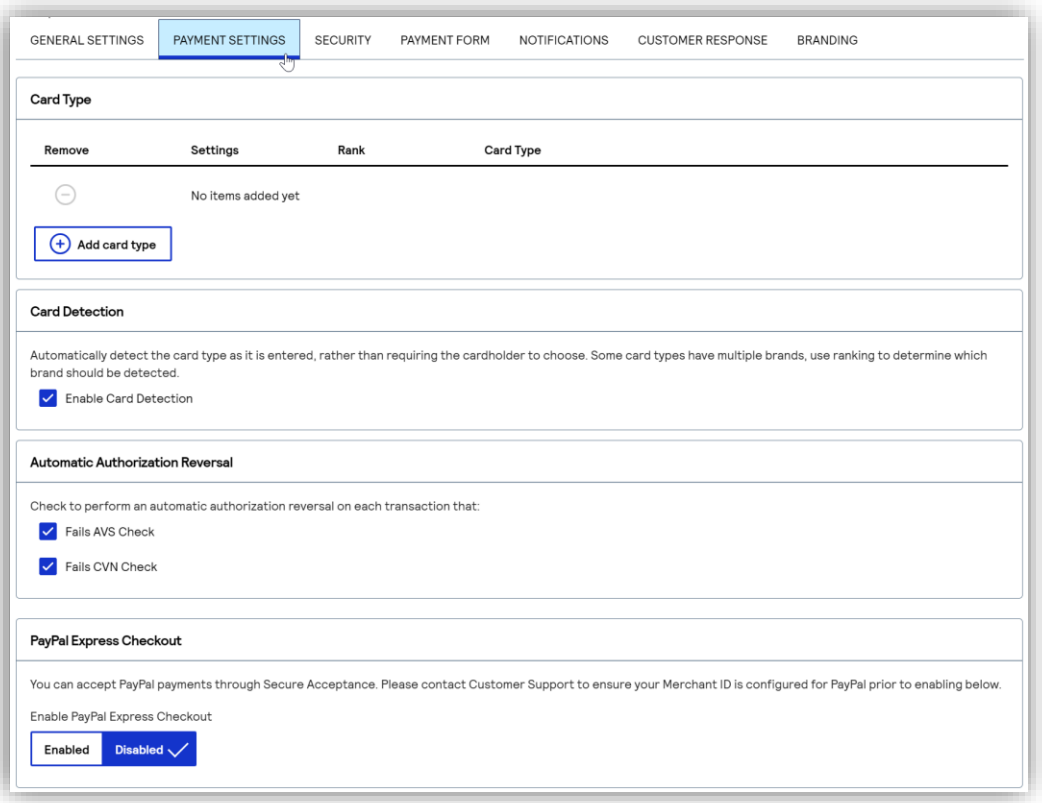
Enter contact info if you choose. At the bottom, under Added Value Services, ensure **Payment Tokenization** is selected. This must be enabled. Decision Manager and BIN Lookup are optional.

Once you've completed this form, click **'Submit'**.

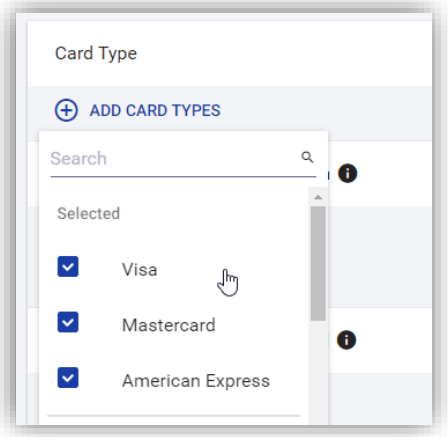


You'll be taken to a new page with a number of tabs, and the info you just entered shown under GENERAL SETTINGS.

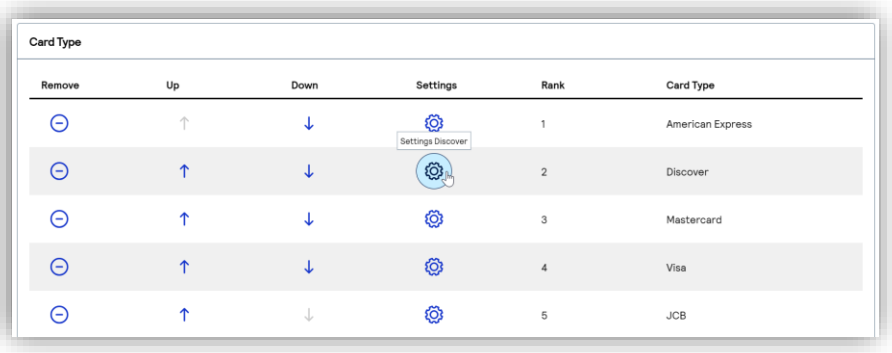
Switch to tab **PAYMENT SETTINGS**.



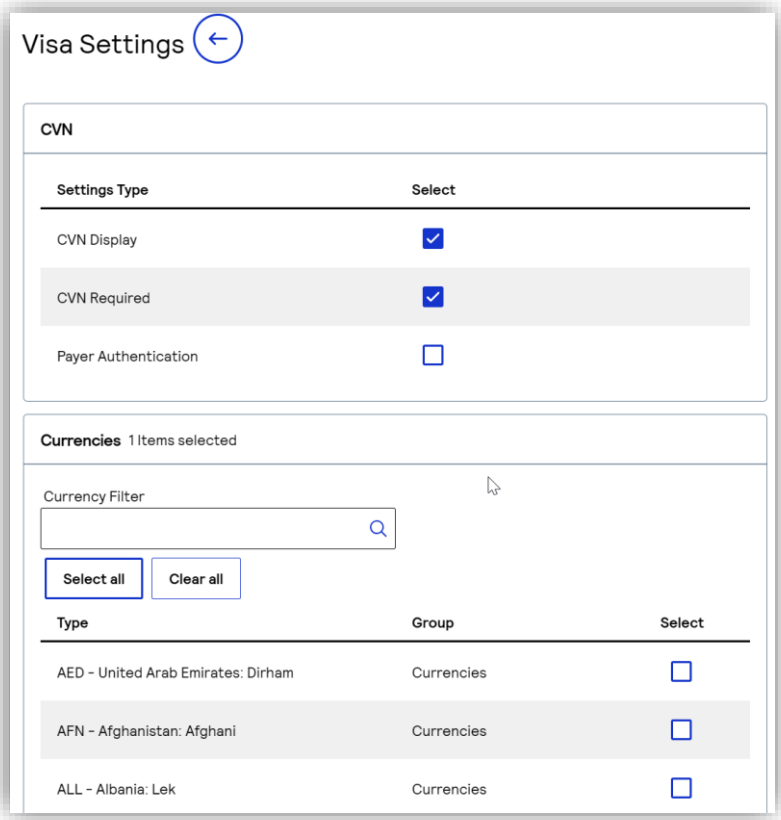
On this tab, you need to configure each card type you want to accept. Click **Add Card Type**, and select those types:



Then click outside the dropdown to close it and apply the changes. You should see all of the selected types now. For each one you selected, you need to click the gear to configure it:

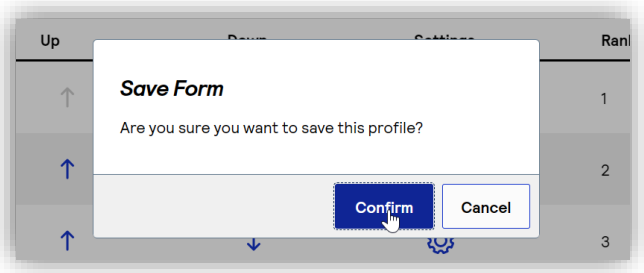


For each type, **enter the settings** and **select CVN display and required** (unless you do not intend to require card CVN entry), and **select your currency(s)**. NOTE: If you have multiple currencies configured in Magento, you MUST enable each currency, for each type. Once you've configured a type, hit **SUBMIT**, then do the next one.

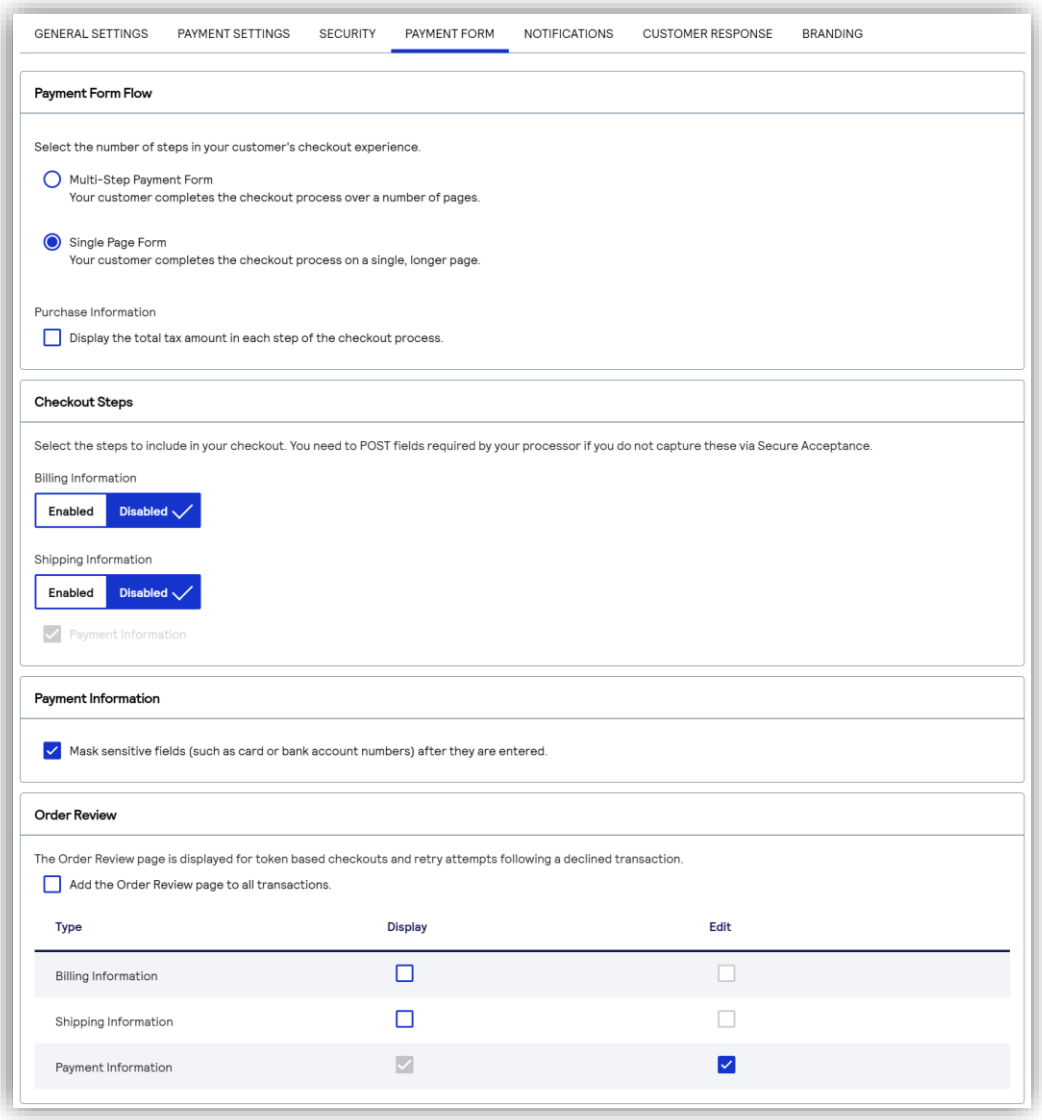


You can ignore the 'Payer Authentication' setting on the popup, and the 'Payer Authentication', 'Automatic Authorization Reversal', and 'PayPal Express Checkout' settings back on the Payment Settings tab. We only use Secure Acceptance for card storage (not for placing actual payment transactions), so none of those settings are applicable.

Once you've added and configured all of your card types, click **SAVE** on the Payment Settings tab.



Now switch to the **PAYMENT FORM** tab (we'll skip Security for now).

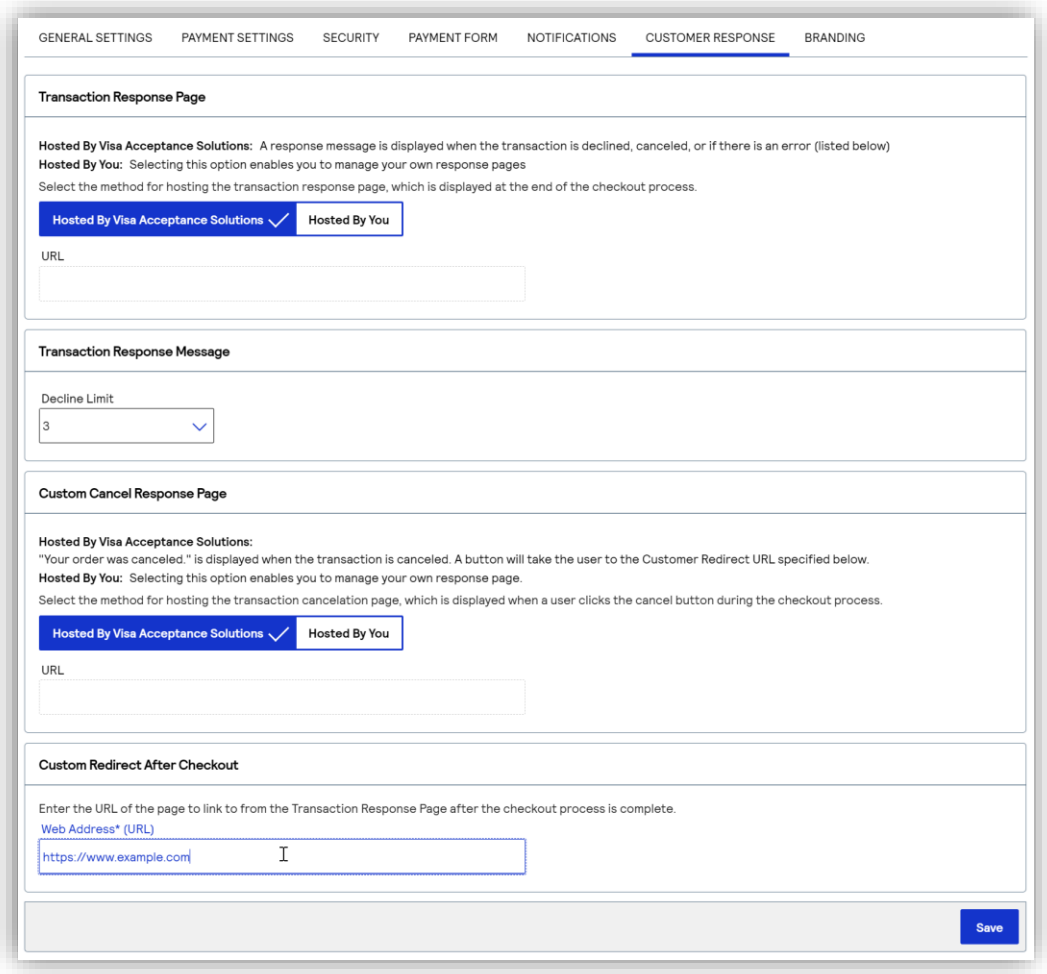


- For Payment Form Flow, select **Single Page Form**. This is required for embedding the form in checkout.
- For Payment Information, select **Mask sensitive fields**. This will hide the CC#/CVN after they're typed in.

- For Order Review, select **Edit** for **Payment Information**. This allows cards to be updated from the customer account.
- Do not select Billing or Shipping under checkout steps or order review. Those are entered on your standard Magento checkout, and including them on the payment form would make it substantially longer.

Once you've set the PAYMENT FORM options to match the above screenshot, hit **SAVE** to apply the changes.

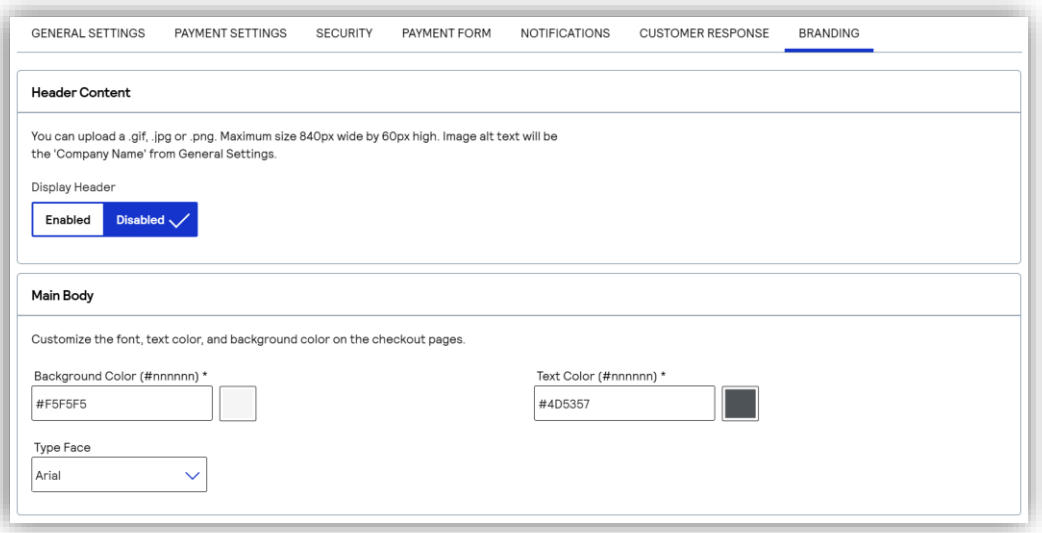
Now switch to the **CUSTOMER RESPONSE** tab:



- For the Transaction Response Page, select **Hosted By Visa Acceptance Solutions**.
- For Transaction Response Message, set the Decline Limit to 5 or less. We suggest 3.
- For the Custom Cancel Response Page, select **Hosted By Visa Acceptance Solutions**.
- For the Web Address* (URL) field at the bottom, enter **your website domain**.

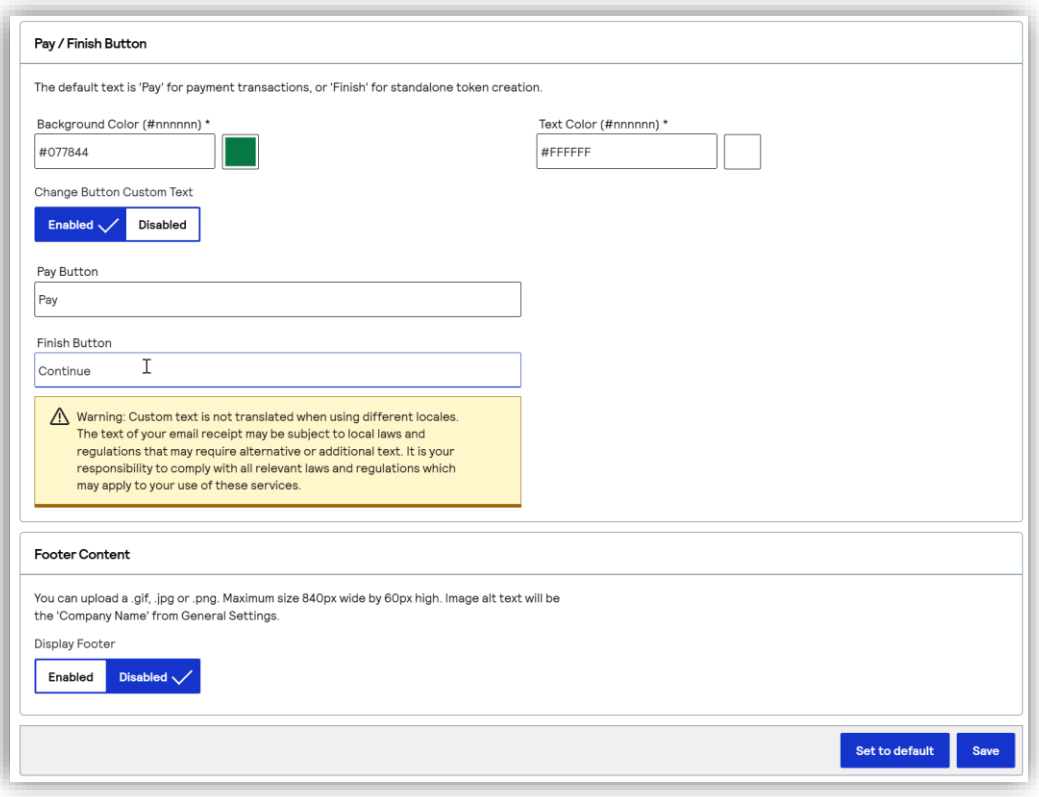
Once you've set the CUSTOMER RESPONSE options to match the above screenshot, hit **SAVE** to apply the changes.

Now switch to the **BRANDING** tab:



This tab changes how the form looks. You should change the colors and font to match your site as much as possible. If nothing else, we suggest changing the **Main Body Background Color** to white (#FFFFFF). Assuming your checkout has a white background, this will display the payment form like it's a seamless part of your checkout form. Otherwise you may be able to see a colored box surrounding the form.

You can ignore the Header Content, Total Amount, Progress Bar, and Footer Content sections. Those don't apply to our extension.

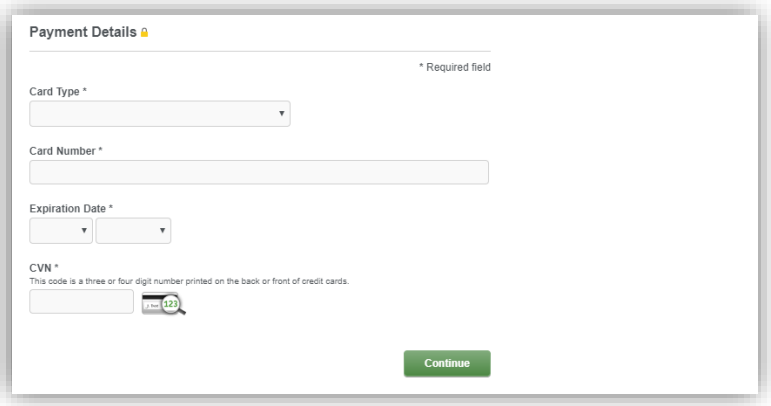


At the bottom of this tab are the Pay / Finish Button, and Button Custom Text.

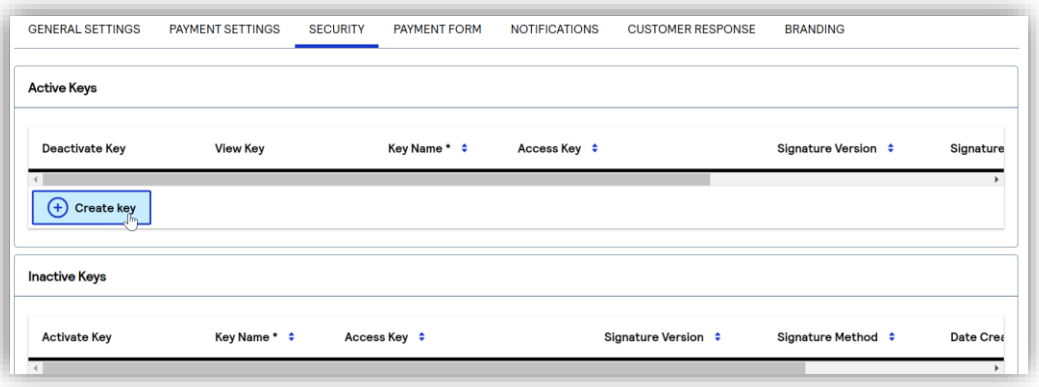
Change the button **Background Color** to match your site branding. By default, the button is green like below.

Finally, set **Change Button Custom Text** to **Enabled**.
Set the Pay Button label to **Pay**.
Set the Finish Button label to **Continue**.

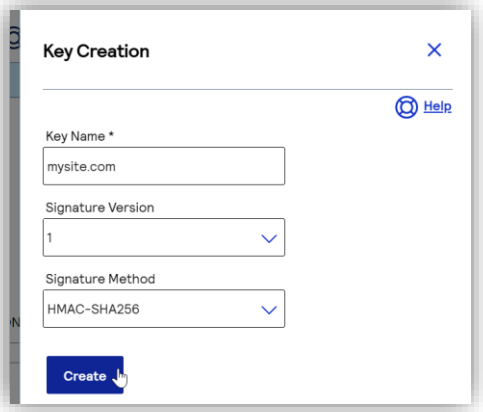
Once you've completed these settings, click **SAVE** to apply the changes.



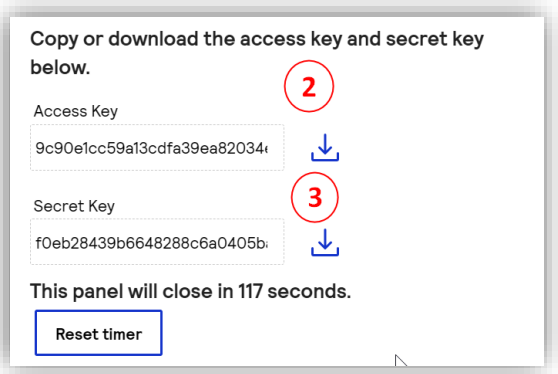
Now switch to the **SECURITY** tab. Click the **Create Key** button on the left side. This will bring up a form.



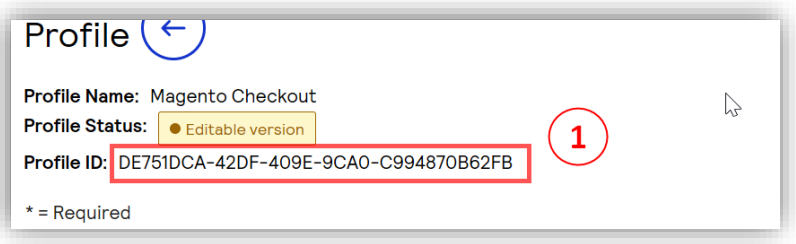
Type in a **Key Name** (any name you'd like), then click **CREATE**.



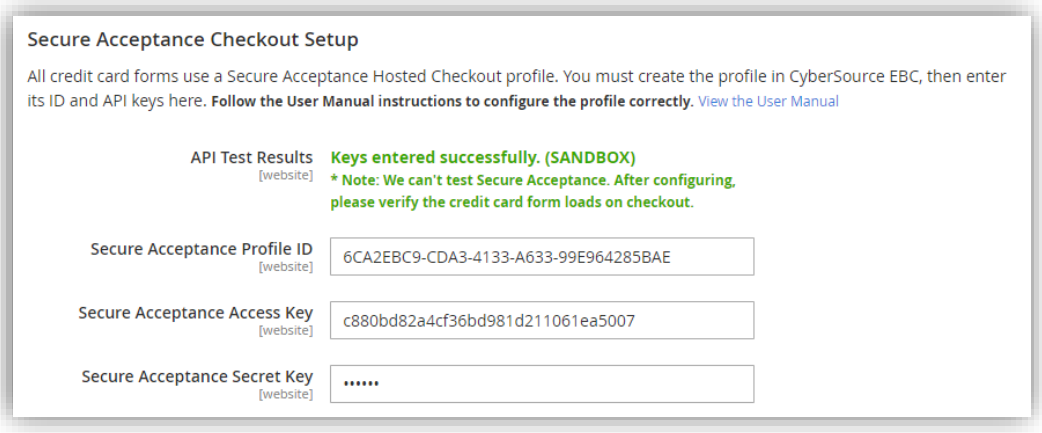
You'll be shown two fields below the form with your **Access Key** and **Secret Key**. Quickly **download or copy these two values**. The form will auto-close after 120 seconds, but you can reopen it if you need to – so don't stress out if you don't get everything.



After the popup has closed, you can find the third API key you need, the **Profile ID**, above the form tabs.



Back on your **Magento Payment Methods Settings** page, go to the **Secure Acceptance Checkout Setup** section.



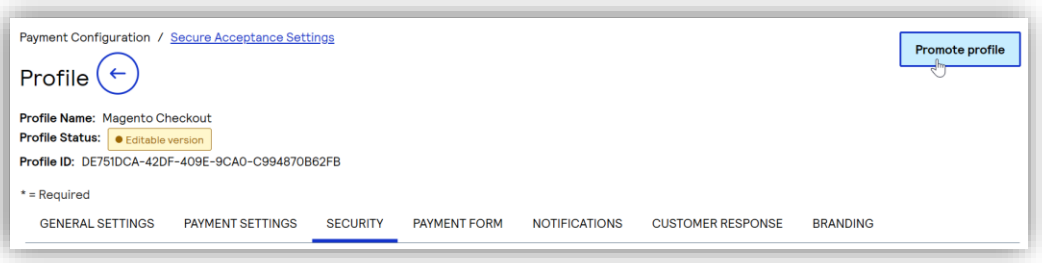
Fill in the Secure Acceptance Profile ID (#1), Access Key (#2), and Secret Key (#3) settings with the values highlighted on the previous screenshots.

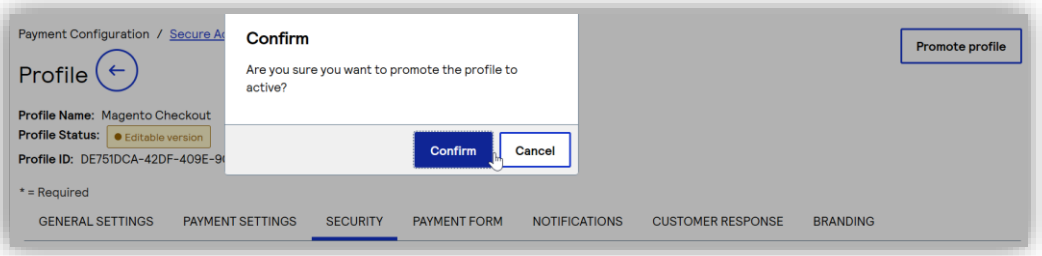
If you didn't get the Access Key or Secret Key, you can select the key and click 'View Key' to see them again:

Once you've entered all three values into Magento, click **Save Config**. Once the page reloads, you should have a green message: **Keys entered successfully.** Note that we can't test these keys, so you will have to actually go to checkout once everything is configured and enabled, to make sure the form loads properly. If you get a red message, verify that you entered the three values correctly and didn't miss any part of them.

FINALLY, back in **CyberSource EBC**, **click 'Promote profile'** at the top right to activate the profile. The checkout form will not work until you've done so.

Make sure you activate ('promote') the profile, or your checkout form will not work.

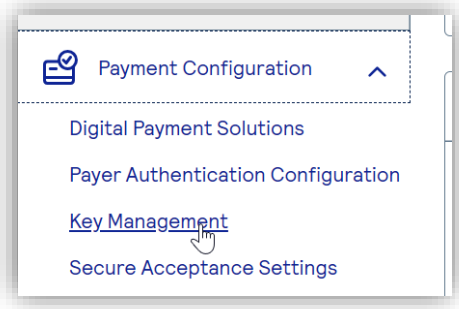




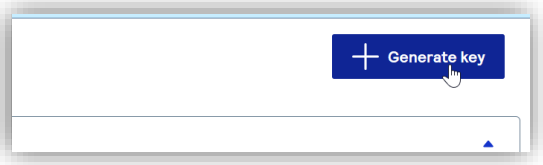
That was the hard part. Almost done.

Step 4. REST API Setup

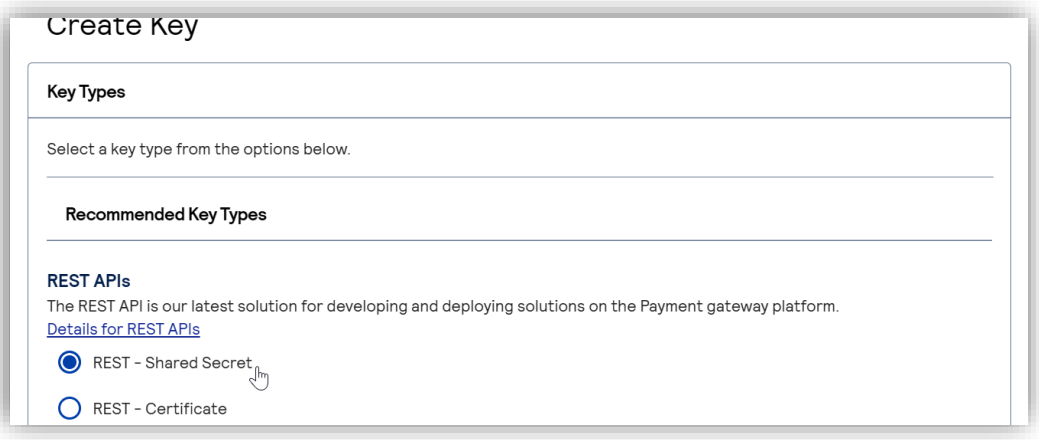
In CyberSource EBC, go back to the **Key Management** section:



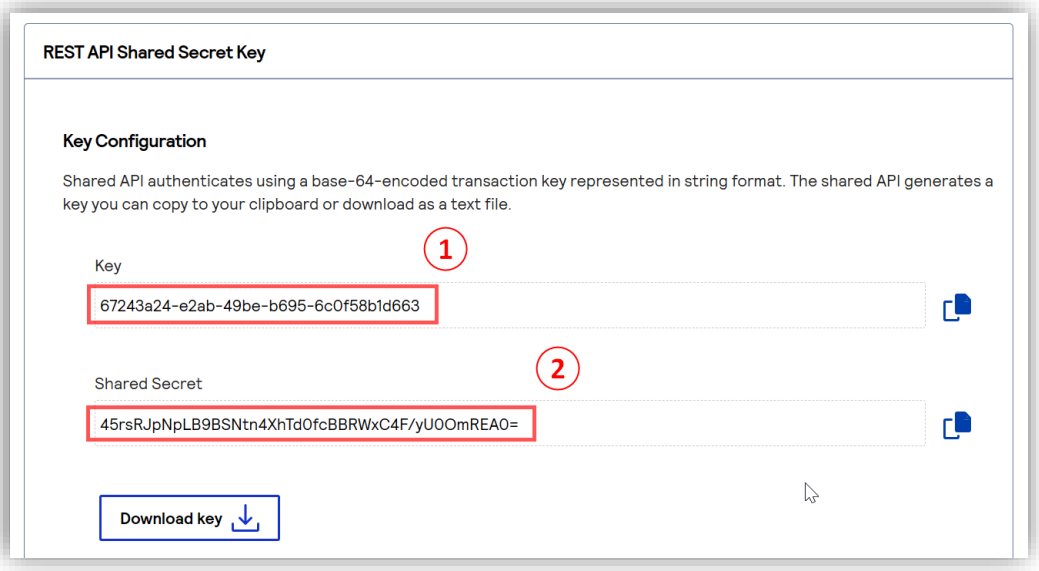
Click **Generate Key**:



Click **REST – Shared Secret**, then **Generate Key**:

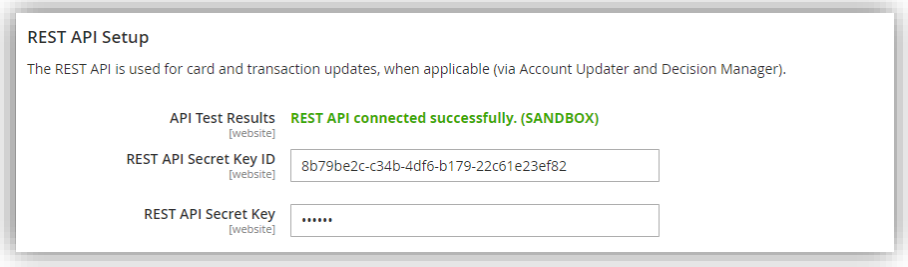


On the next page, it will show the generated API keys.



Copy the **Key** (#1) into the **REST API Secret Key ID** field, and the **Shared Secret** (#2) into the **REST API Secret Key** field in Magento.

You should have both REST API fields completed now in Magento. **Save Config** to save the changes and test.

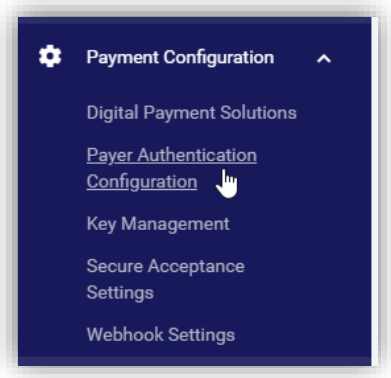


If you did everything in this step correctly, after saving you should see a green message in the REST API Test Results: **REST API connected successfully.** If you get a red message, fix the described error (if any) or verify that you entered the two values correctly and in their entirety.

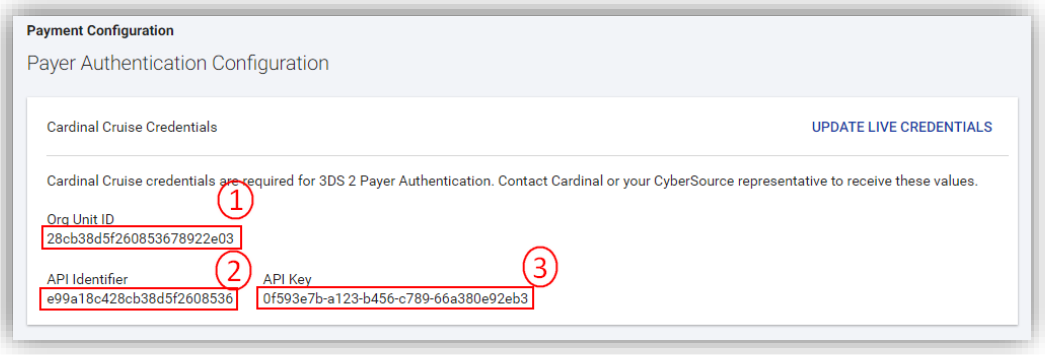
Step 5. Payer Authentication (3D Secure) Setup

Payer Authentication is optional. If you don't want to use 3D Secure, you can skip this. Just set 'Enable Payer Authentication' to 'No' in Magento. Otherwise:

This one is easy. In EBC, go to **Menu**, then **Payment Configuration**, then **Payer Authentication Configuration**.

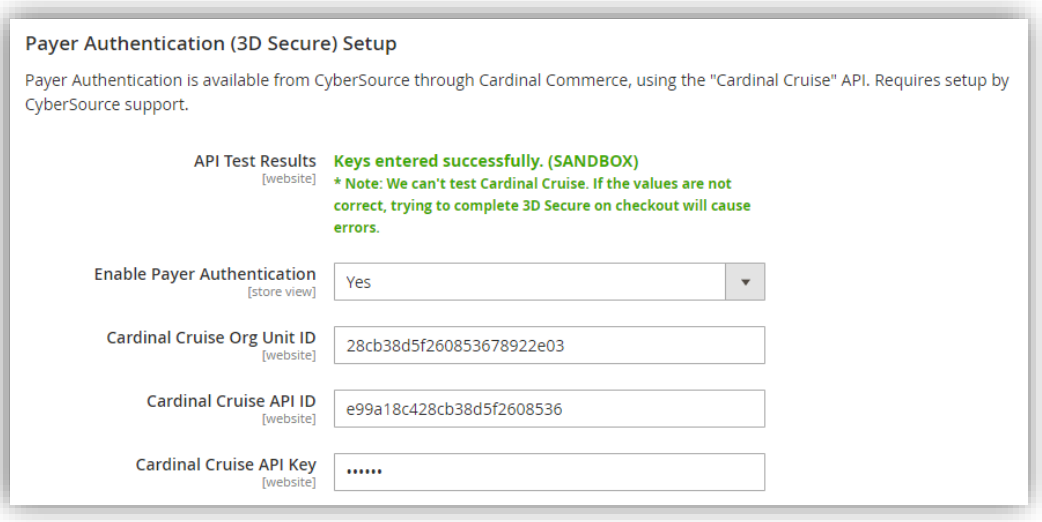


If you've set your account up for Payer Authentication with CyberSource, you should see three credentials on this page.



Copy these values into Magento as your **Cardinal Cruise Org Unit ID** (#1), **Cardinal Cruise API ID** (#2) and **Cardinal Cruise API Key** (#3).

If your account does not have any Cardinal Cruise API credentials yet, please contact your CyberSource representative to receive the values.



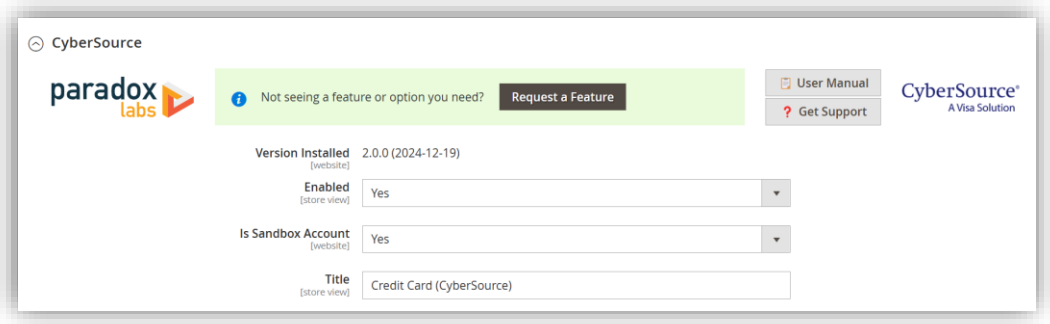
Once you've entered the values into Magento, click **Save Config** to save and verify the settings. If it's green then the values are in the expected format. Note that we can't test to verify they are actually correct and valid, so be careful to ensure they match the Cardinal Cruise Credentials you were given. If the values are not correct, customers may be unable to place orders on checkout.

Congratulations! You've completed connecting your new Magento extension to your CyberSource account. Please see the next section for information on the rest of the configuration options in Magento.

Configuration

Open your Admin Panel and go to **Admin > Stores > Settings > Configuration > Sales > Payment Methods**. Toward the bottom of the page, you'll find a 'CyberSource' settings section like the below.

General

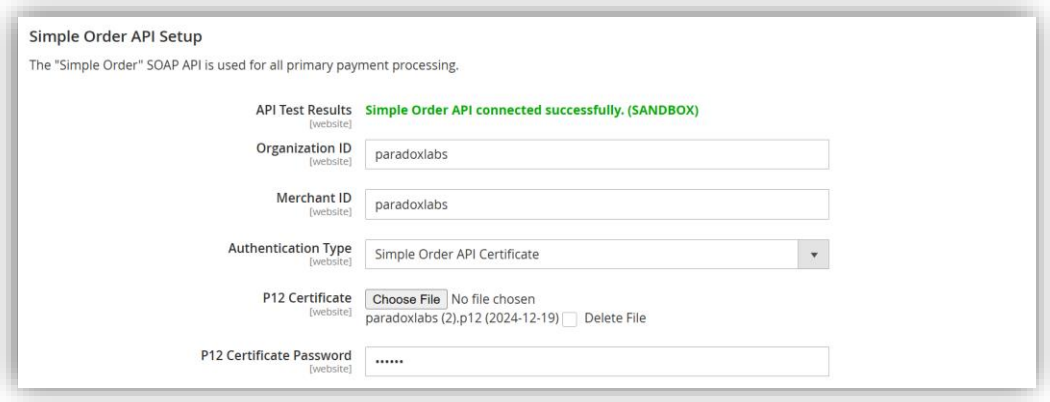


- **Version Installed**: This tells you the version of our extension currently installed on your website. Please include this in any support requests.
- **Enable**: Yes to enable the payment method. If disabled, you will still be able to invoice/refund existing orders, but it will not show up as a payment option during checkout.
- **Is Sandbox Account**: If Yes, all requests will be made to CyberSource's test APIs, and no actual payments will be processed. If you want to test, you must have a sandbox account (separate from your production CyberSource account). You can create one at: https://ebc2.cybersource.com/ebc2/registration/external
- **Title**: This controls the payment option label on checkout and order status pages.

Simple Order API Setup

The "Simple Order" SOAP API connects your store to CyberSource for primary payment processor: Authorizations, captures, refunds, etc. You won't be able to process payments without it.

Please see Connecting A New CyberSource Account: Step 2. Simple Order API Setup for details on finding and configuring these API credentials.



- **API Test Results**: Once you've completed and saved these settings, we will connect to CyberSource to verify that the connection works successfully. If we cannot connect to CyberSource, or your credentials are incorrect, this will tell you with a red message. Correct the error, then reload the page and it should show **Simple Order API connected successfully.**

Secure Acceptance Checkout Setup

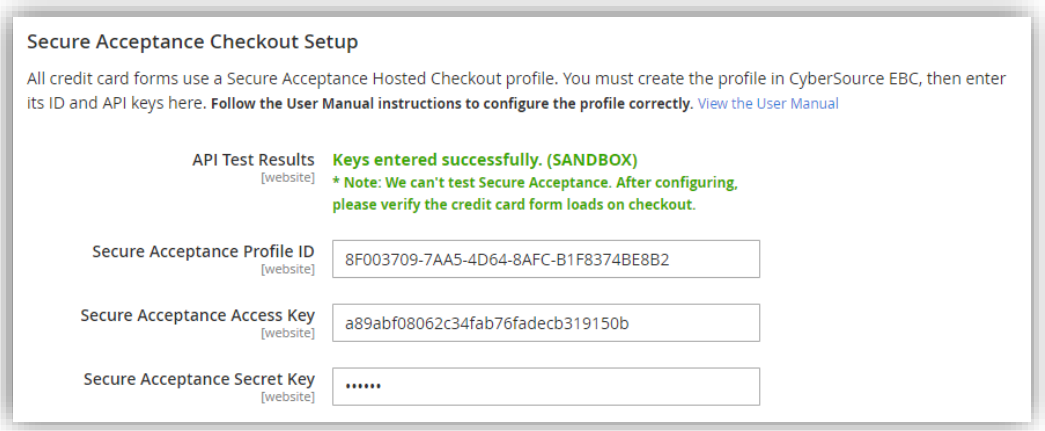
Secure Acceptance Hosted Checkout is used for all payment forms that are part of the extension. The forms load an inline page (iframe) from CyberSource containing the actual payment fields, and when they enter and submit their credit card details, that info gets sent directly to CyberSource for storage. At no point will a credit card number touch your website as part of this extension.

Secure Acceptance requires extensive configuration within CyberSource to set the particular form fields and styles.

If your credit card forms don't load correctly, verify these credentials are correct.

If your credit card forms don't look right, verify that your Secure Acceptance profile settings in CyberSource match the settings we recommend, or update the Branding settings as needed.

Please see Connecting A New CyberSource Account: Step 3. Secure Acceptance Checkout Setup for details on finding and configuring these API credentials.

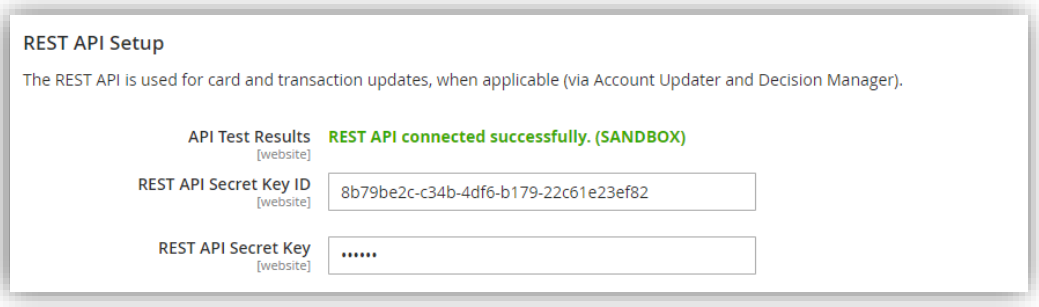


- **API Test Results**: Once you've completed and saved these settings, we verify that they are in the correct format. If they are not as expected, this will tell you with a red message. Correct the error, then reload the page and it should show **Keys entered successfully.** Note that we cannot test to ensure the Secure Acceptance API keys are actually valid. Please open the credit card form on your checkout to confirm it works as expected after changing these values.

REST API Setup

The CyberSource REST API is used for card and transaction updates. If your CyberSource account has Account Updater or Decision Manager, this will sync updates from those into Magento. It may also be used for additional functionality over time.

Please see Connecting A New CyberSource Account: Step 4. REST API Setup for details on finding and configuring these API credentials.

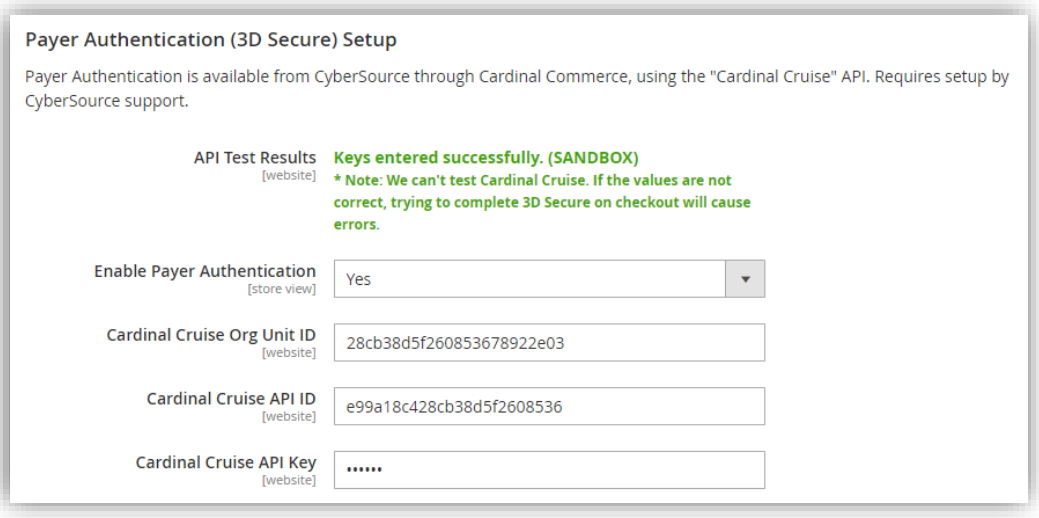


- **API Test Results**: Once you've completed and saved these settings, we will connect to CyberSource to verify that the connection works successfully. If we cannot connect to CyberSource, or your credentials are incorrect, this will tell you with a red message. Correct the error, then reload the page and it should show **REST API connected successfully.**

Payer Authentication (3D Secure) Setup

The Payer Authentication API credentials are used for implementing 3D Secure via Cardinal Commerce's Cardinal Cruise API. If enabled, customers enrolled in 3D Secure programs may be asked to authenticate with their card processor when placing an order. This is mandatory for businesses in Europe to comply with Payment Services Directive 2 (PSD2).

Please see Connecting A New CyberSource Account: Step 5. Payer Authentication (3D Secure) Setup for details on finding and configuring these API credentials.

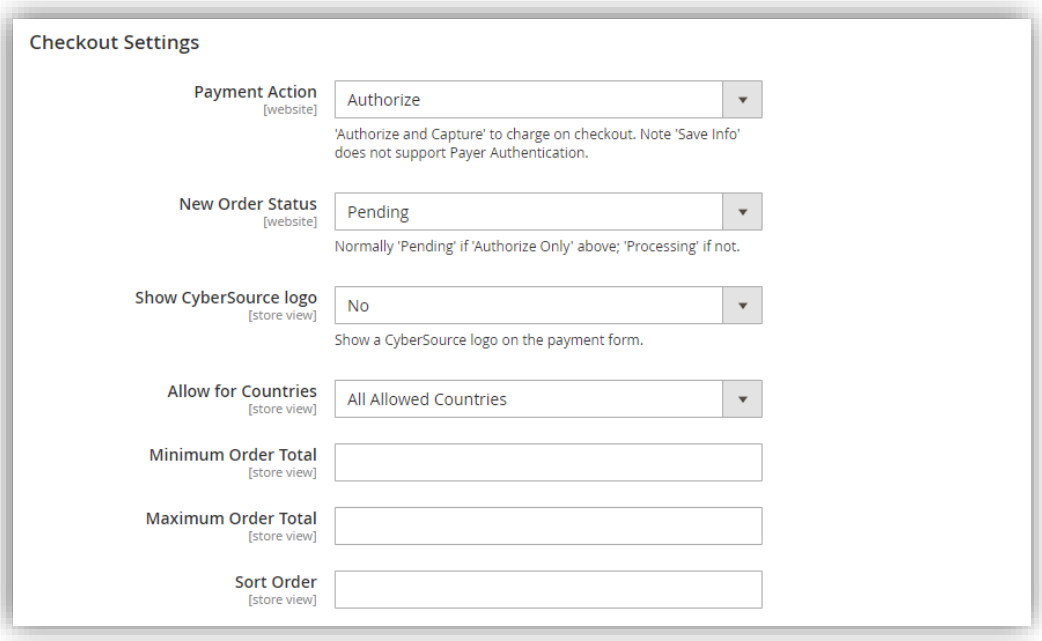


- **Enable Payer Authentication**: If yes, each transaction on frontend checkout will check whether the user's bank requires them to complete the 3D Secure authentication process (in which case they will be shown a

popup from their credit card bank with directions). If set to 'No', no 3D Secure enrollment or verification will be performed.

- **API Test Results**: Once you've completed and saved these settings, we verify that they are in the correct format. If they are not as expected, this will tell you with a red message. Correct the error, then reload the page and it should show **Keys entered successfully.** Note that we cannot test to ensure the Payer Authentication API keys are actually valid.

Checkout Settings

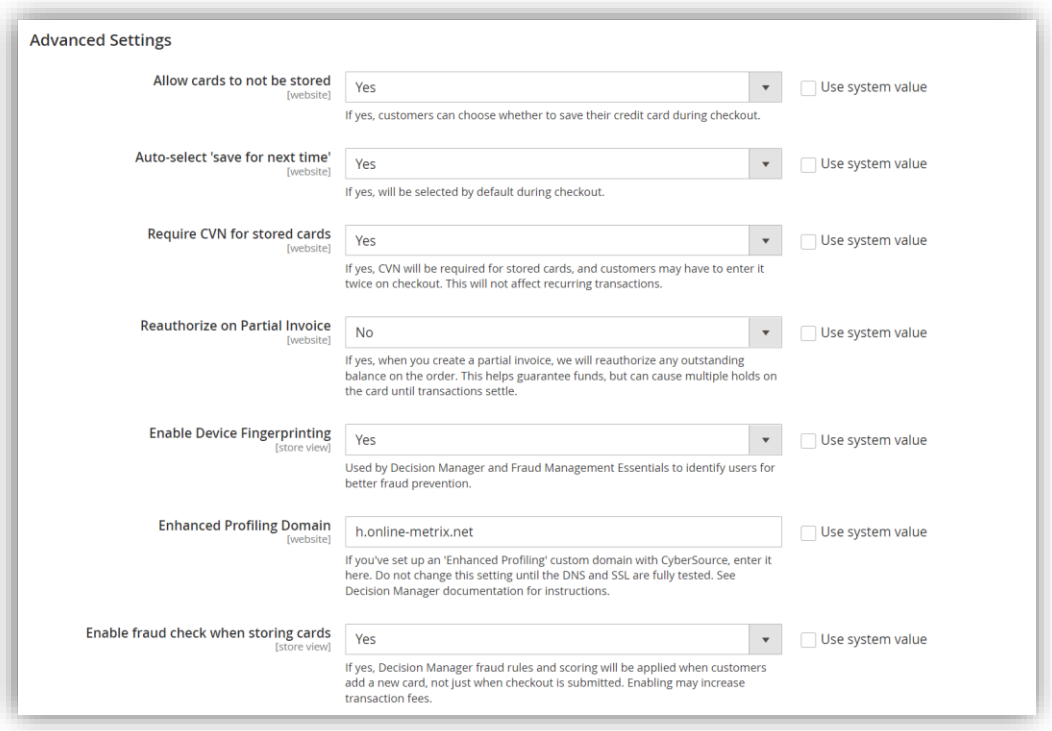


- **Payment Action**: Choose from the following options.
    - **Save info (do not authorize)**: This will require customers to enter a credit card on checkout, and store that credit card in CyberSource. The credit card will be validated in the process. No funds will be captured or held from the credit card upon checkout. Invoicing the order will perform a standalone authorize+capture transaction, but is not guaranteed to go through (funds may not be available). **Note, this is not compatible with Payer Authentication, which requires a transaction be run immediately.**
    - **Authorize**: This will authorize the order amount upon checkout, allowing for manual invoicing and capture of the funds later. The authorized funds will be held (reserved) for about a week depending on your processor. If you do not invoice within that time, the authorization will expire, and invoicing will perform a standalone authorize+capture transaction instead (which is not guaranteed to go through).
    - **Authorize and Capture**: This will capture all funds immediately when an order is placed.

  Payment processors strongly recommend not capturing funds unless/until you are within three days of fulfilling/shipping the order
- **New Order Status**: Set this to your desired initial status for orders paid via CyberSource. Default Magento behavior is 'Pending' for Authorize Only, and 'Processing' for Authorize and Capture.

- **Show CyberSource logo**: If yes, checkout will display a 'CyberSource' logo above the payment form.
- **Allowed for Countries**: This setting allows you to limit which countries are allowed to use this payment method.
- **Minimum Order Total**: This setting allows you to set a minimum order value for the payment option. For instance, set to 5 to only allow credit card checkout for orders of $5 or more.
- **Maximum Order Total**: This setting allows you to set a maximum order value for the payment option. For instance, set to 1000 to only allow credit card checkout for orders of $1000 or less.
- **Sort Order**: This setting allows you to change the order of payment options on checkout. Enter a number for this and all other payment methods according to the order you want them to display in.

Advanced Settings



- **Allow cards to not be stored**: If yes, customers will have a 'Save for next time' checkbox on checkout. If no, logged in customers will see a message instead: *"For your convenience, this data will be stored securely by our payment processor."* Guests will never be given the option to store a credit card.
  Note that all cards are always stored in CyberSource, regardless of this setting or the customer's choice. This is necessary for payment processing. If the order is placed as a guest, or the customer chooses to not save their card, it will be automatically purged from all systems 120 days after its last use. This ensures the info is available for edits, captures, and refunds, but respects the customer' wishes.
  If a card is 'not saved', it will not display under the customer's saved credit cards (Account > My Payment Data), nor will it be selectable during checkout. Note that as an admin, order 'edit' or 'reorder' will bypass this, always allowing reuse of the original payment info (unless it was since purged).
- **Auto-select 'save for next time'**: If yes, the 'save this card for next time' checkbox will be checked by default. If no, customers will have to explicitly select it to store and reuse their card.

- **Require CVV for stored cards**: If yes, customers and admins will be required to enter the credit card CVV every time a card is used at checkout. Note enabling this will require the CVV to be entered twice when adding a new card—once when storing the card with CyberSource, and again when placing the actual order.
- **Reauthorize on Partial Invoice**: If yes, and you invoice part of an order, a new authorization will be created for the outstanding order balance (if any). This helps guarantee funds. Any failure during reauthorization is ignored.
- **Enable Device Fingerprinting**: Enable if you use Decision Manager or Fraud Management Essentials in CyberSource. This includes a script that records details about the user's browser for identification purposes, for fraud prevention. By default this script is loaded from "h.online-metrix.net".
  Enabling this may result in third-party cookies being included during your checkout process. Please view the *Decision Manager Device Fingerprinting Guide, Appending C: Device Fingerprinting Cookie FAQ*, if you need more information. This guide is available through *CyberSource EBC > Menu > Fraud Management > Guides*.
- **Enhanced Profiling Domain**: If you use Device Fingerprinting, you can use this to set a custom 'Enhanced Profiling' domain for the script to load it from your domain rather than "online-metrix.net". This requires special setup in coordination with CyberSource. Please see the *Decision Manager Device Fingerprinting Guide, chapter 1, section "Enhanced Profiling"*, for more information. This guide is available through *CyberSource EBC > Menu > Fraud Management > Guides*.
- **Enable fraud check when storing cards**: If yes, cards will be validated and processed through your fraud rules at the time of storage on checkout. Otherwise, fraud rules will not be applied until time of the actual order transaction when placing an order. Enabling this setting may increase your payment fees by running an additional transaction; this behavior is disabled by default.

This concludes the payment method's configuration options.

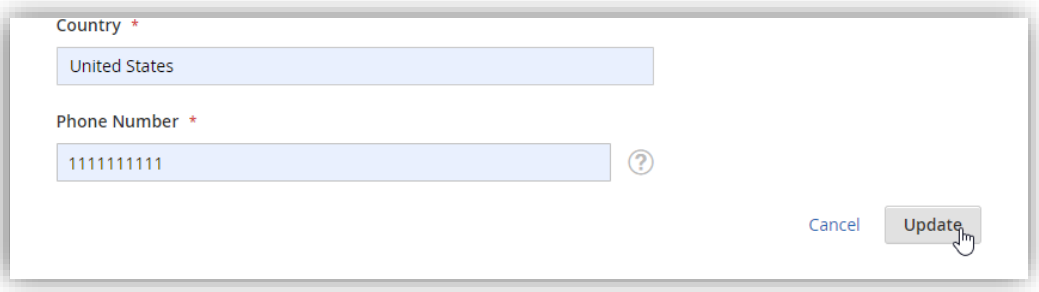
Behavior Notes

For the most part this is a standard Magento payment method, with the addition of the advanced Stored Card functionality and everything related to that. However, there are some things worth note:

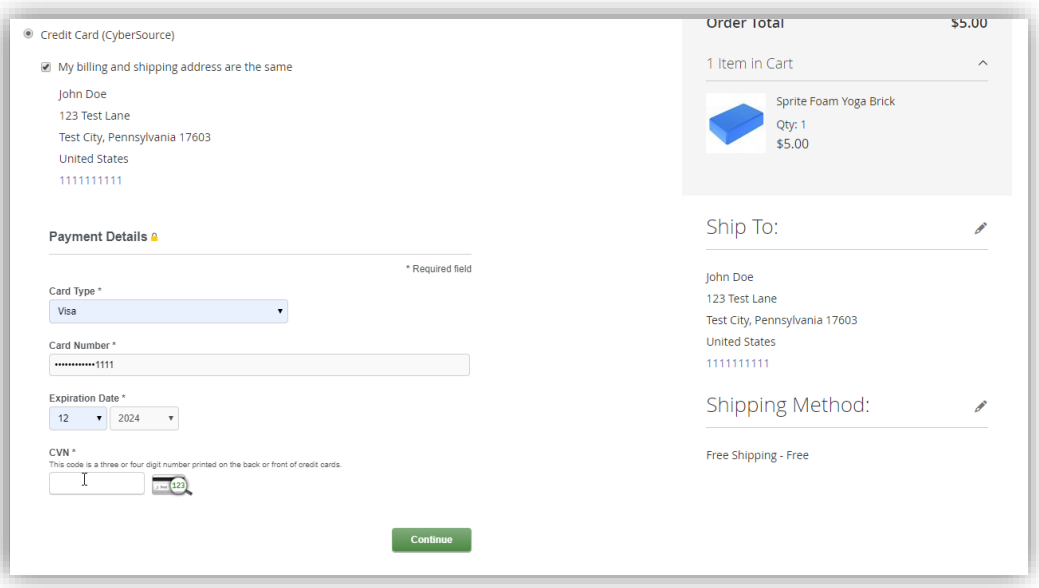
User Experience

One difference from typical Magento payment methods is that we use CyberSource's secure iframe solution, Secure Accept Hosted Checkout. This allows customers to enter all of their CC info directly to CyberSource. It does make for a slightly different payment flow from a typical payment method, though:

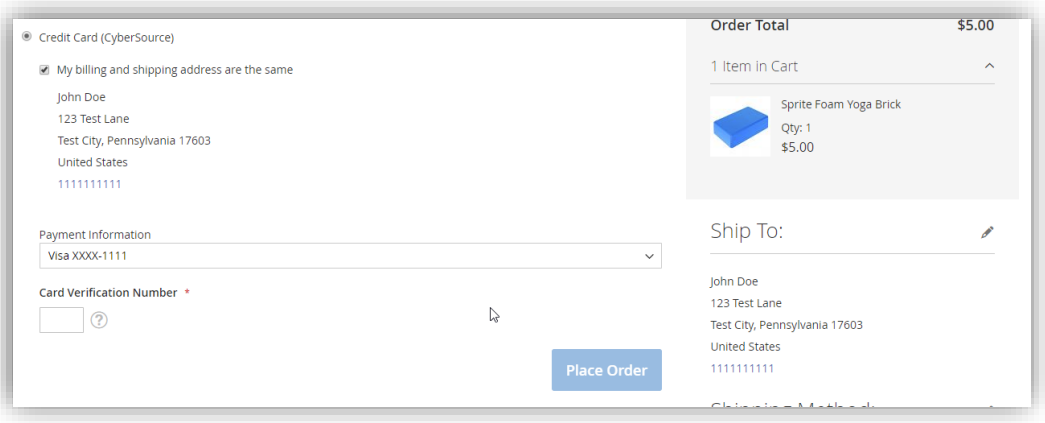
Customers must always enter and confirm their billing address first. This may default to "My billing and shipping address are the same", in which case the payment form will auto-load, but if not they'll need to enter and 'Update' their address first.



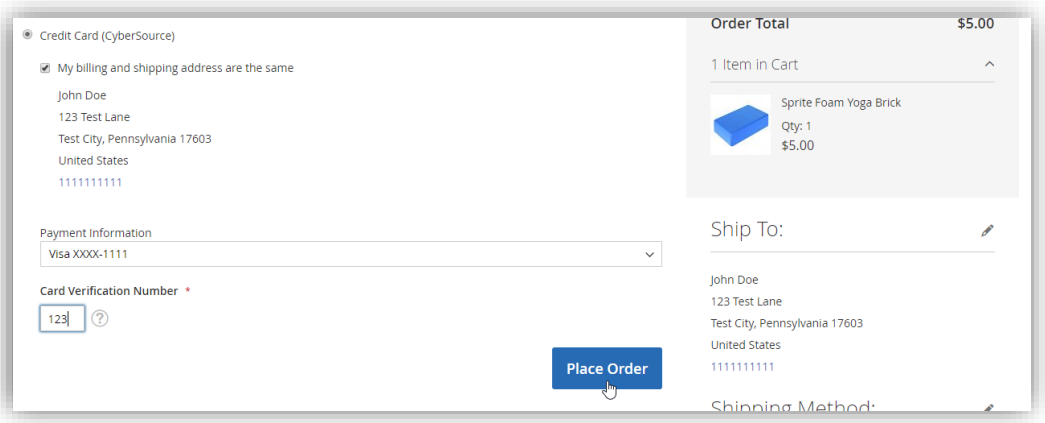
After confirming billing address, the payment form will load:



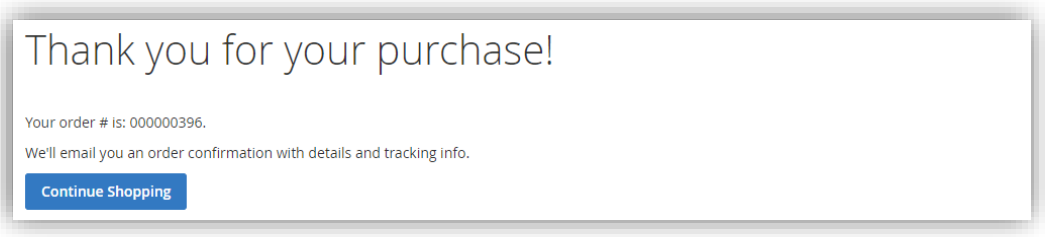
Complete the form and hit 'Continue' to store the information, and the entered card will be shown preselected in a dropdown:



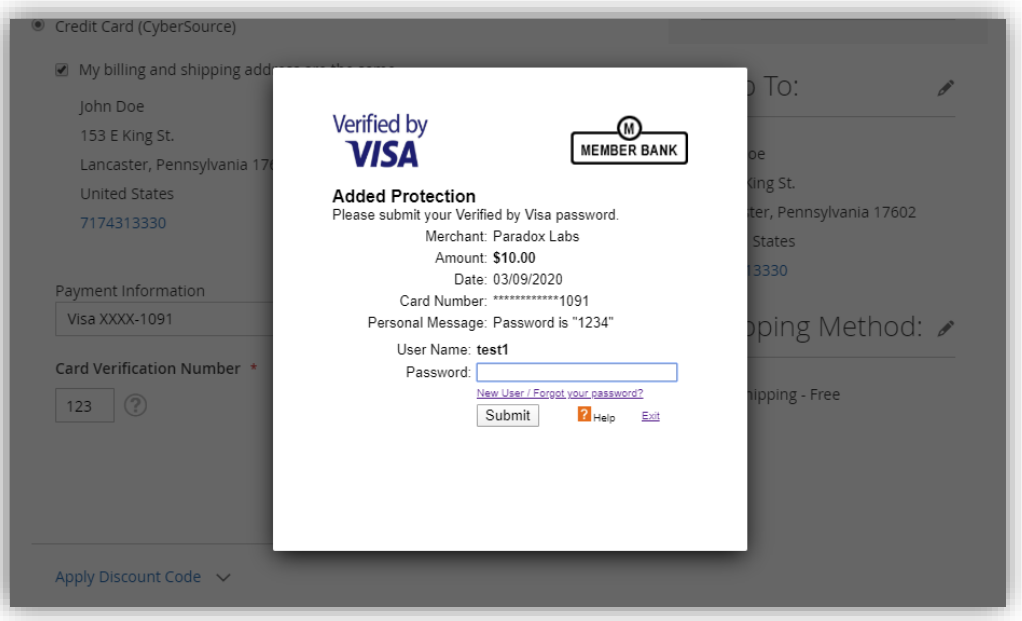
If you require CVN for stored cards, the customer will need to re-enter the code before they can place the order.



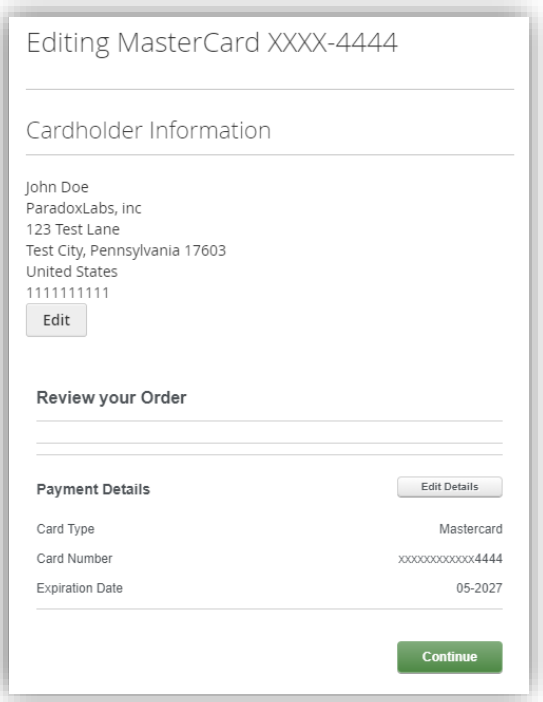
Then hit Place Order, and that's it!



If you have Payer Authentication (3D Secure) enabled, when the user hits place order they may be required to authenticate. If so, they will be shown a popup on top of your checkout page. The exact display and requirements of the popup may vary by credit card and enrollment type. Once the user completes the process, checkout will automatically resubmit.



Card management within a customer's account (on frontend or admin) follows a similar flow, requiring address confirmation before entering or updating payment information. When editing a stored card, the expiration date can be updated without re-entering the full credit card number—but CVN will always be required.



Note that when editing an existing stored card, the form will show a "Review your Order" header above the payment details summary – this is unfortunately an unavoidable part of the Secure Acceptance forms.

Security

Secure Acceptance Hosted Checkout will always be active. There is no option to turn it off and revert to 'normal' credit card input. All credit card details (CC number, expiration date, CCV) are entered into an iframe hosted by CyberSource, and transmitted directly to CyberSource's servers. At no time will credit card numbers ever touch your server for this payment method. This allows for the best possible mix of user experience and PCI compliance.

Some data about credit cards is kept in your database. This includes the card type, last 4 digits, expiration date, and identifiers, among other data. All of this data is available via the CyberSource API, and no part of it is considered 'Confidential Cardholder Data' in the context of PCI compliance.

If you intend to collect payments from other sources using the Magento API, you will need to use a CyberSource API to tokenize the credit card, then store that card and token in Magento as a TokenBase Card, via the REST or GraphQL API. See the Technical / Integration Details section at the end for more information. This payment method is incapable of processing a transaction or storing a card using raw credit card details; it will not accept them under any circumstances.

3D Secure / Payer Authentication

3D Secure applies to standard frontend checkout only. It is not currently compatible with multishipping checkout -- if you enable 3D Secure, the CyberSource payment method will be unavailable for multishipping checkout. It also does not apply to any admin or internal transactions, as 3D Secure requires the user's direct involvement (and their authentication password must not be shared with any other persons under any circumstances). Admin transactions will be run with a 'Mail or Telephone Order' commerce indicator instead.

3D Secure is implemented via Cardinal Commerce's Cardinal Cruise Hybrid JavaScript API, which works hand in hand with the CyberSource Simple Order API. This provides complete 3D Secure 2 support, and should automatically support new 3D Secure 2 requirements over time.

Card Storage

Credit cards will always be stored in CyberSource Token Management Service to allow advanced functionality, even if the customer chooses not to save upon checkout. If the customer chooses not to save, or is a guest, the card will not show up for reuse within Magento. In this case, the card will be automatically purged from all systems after 120 days past its last use. This ensures the info is available for edits, captures, and refunds, but respects the customer' wishes.

There is no automatic card duplicate prevention: If a customer enters the same credit card as a new card 3 times, it will be stored within CyberSource 3 times. There is no limit to the number of cards/tokens on a CyberSource account.

Account Updater

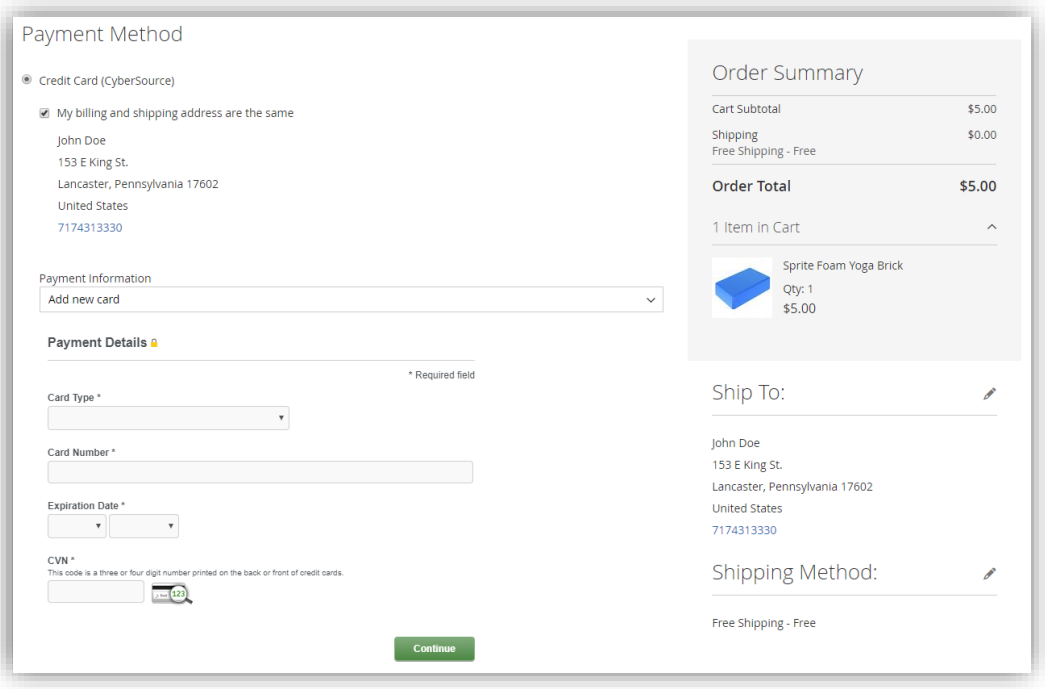
Account Updater is supported, and automatically active if you configure REST API credentials and have Account Updater enabled on your CyberSource account itself. There is a charge associated with using CyberSource's Account Updater service. When enabled with CyberSource, your stored cards within Magento will automatically update to reflect changes in expiration dates, card numbers, and closed accounts.

Usage

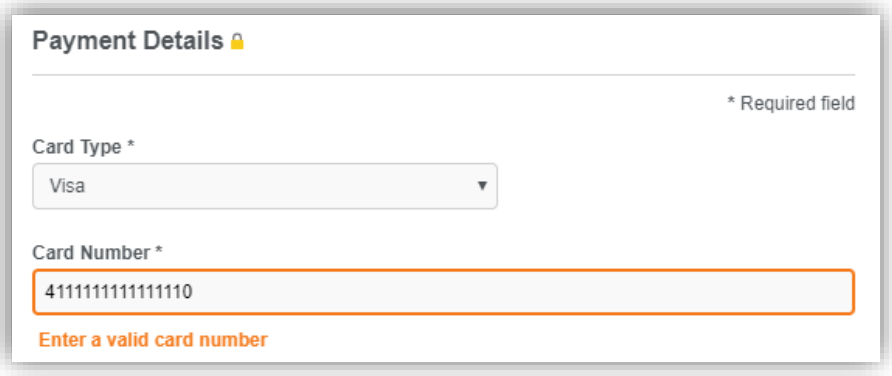
There isn't much to using this extension in practice: It's a standard Magento payment method, and all interfaces should be self-explanatory. Here's a rundown of what you get:

Checkout Payment Form

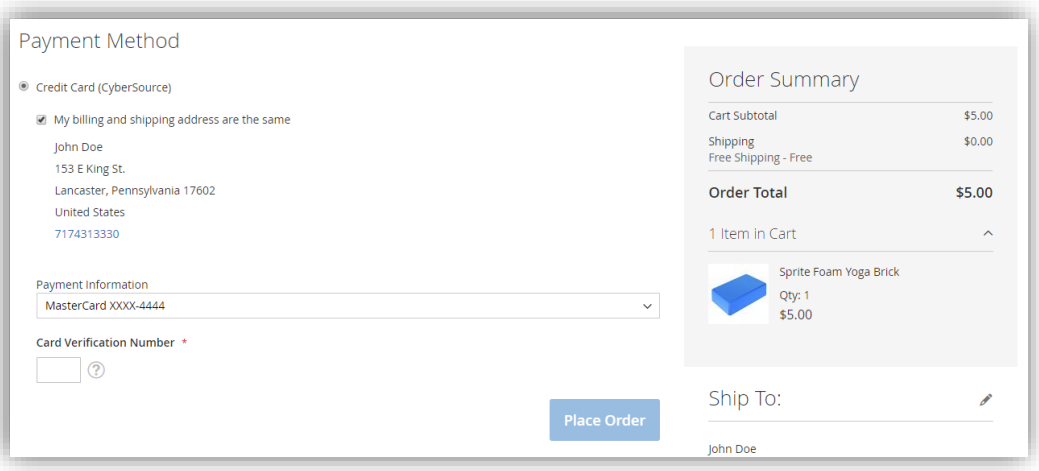
The frontend payment form lets you choose/enter billing address and credit card. You can choose an existing card (if any) from the dropdown, or to add a new one.



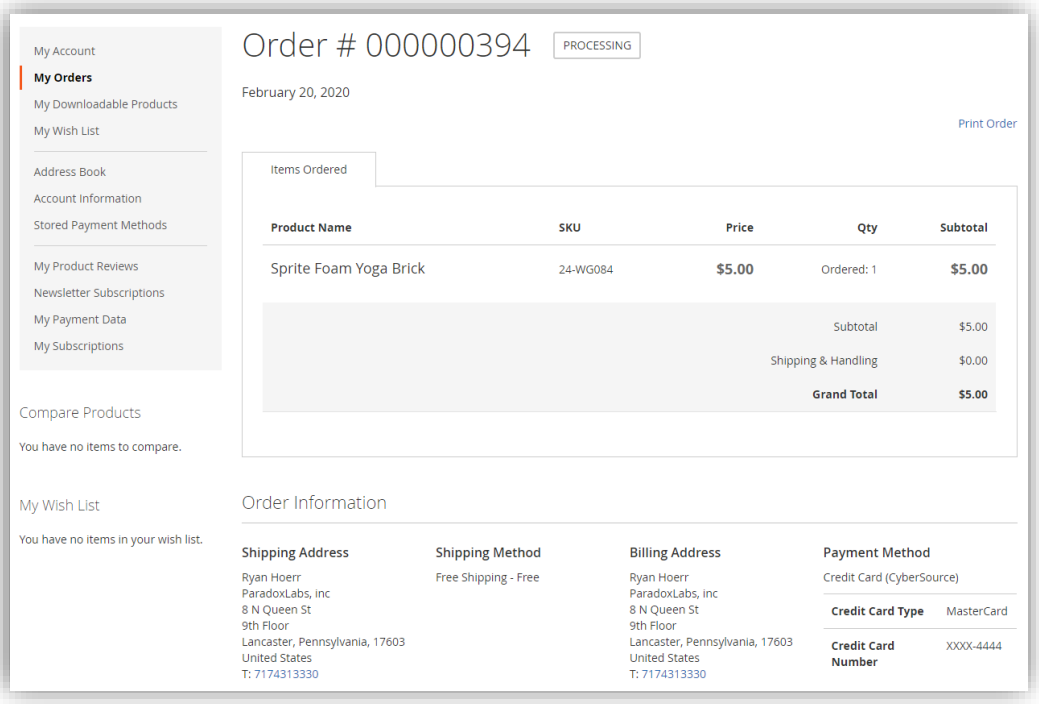
Any format errors are immediately displayed below the input field.



If the customer has stored cards, their most recent one will be selected by default:

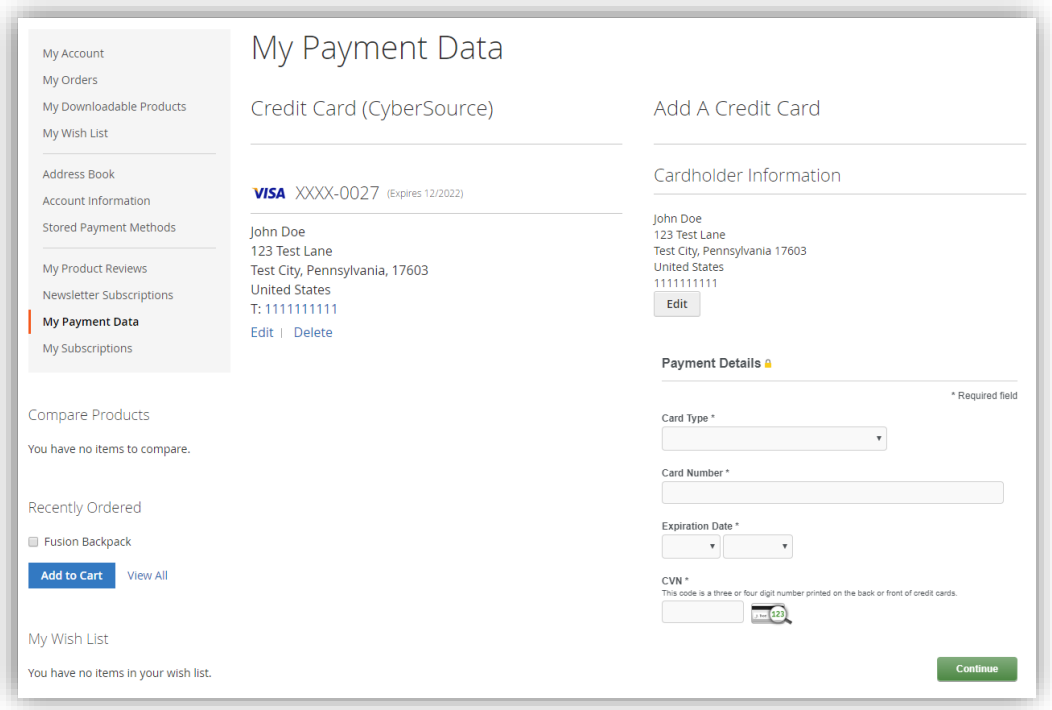


Order status page



Customer ‘My Payment Data’ account area

The My Payment Data section allows customers to see their stored cards, add, edit, and delete.



Like on checkout, the address must be entered and confirmed before payment info can be entered. This means all card adding/editing is a two step process. You can go back and edit the address after confirming it, but that will clear out any changes that may have been made to payment data.

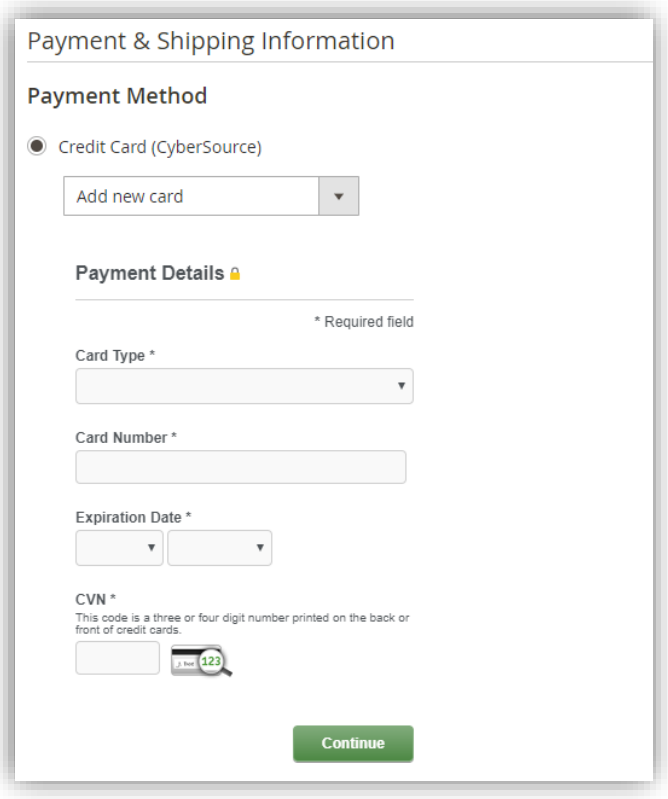
When editing a card, you can edit the address without changing the card details, or the card details without changing the address, or update the expiration date without reentering the card number. Any change to the card details will require the CVN to be reentered, if CVN is enabled.

Note that cards associated with an open (uncaptured) order cannot be edited or deleted. They will display a *‘Card In Use’* message in place of the buttons. As soon as all orders paid by the card are completed, the ‘Edit’ and ‘Delete’ buttons will appear.

To prevent abuse, the Payment Data section will only be available to customers after they have placed a successful order. If a customer attempts to access the page before then, they’ll be redirected to the Account Dashboard with the message, *“My Payment Data will be available after you've placed an order.”* Also to prevent abuse, if a customer receives errors trying to save a card five times, they will be blocked from access for 24 hours with the message, *“My Payment Data is currently unavailable. Please try again later.”* Both of these behaviors can be adjusted or disabled if necessary; please contact us if you have a problem.

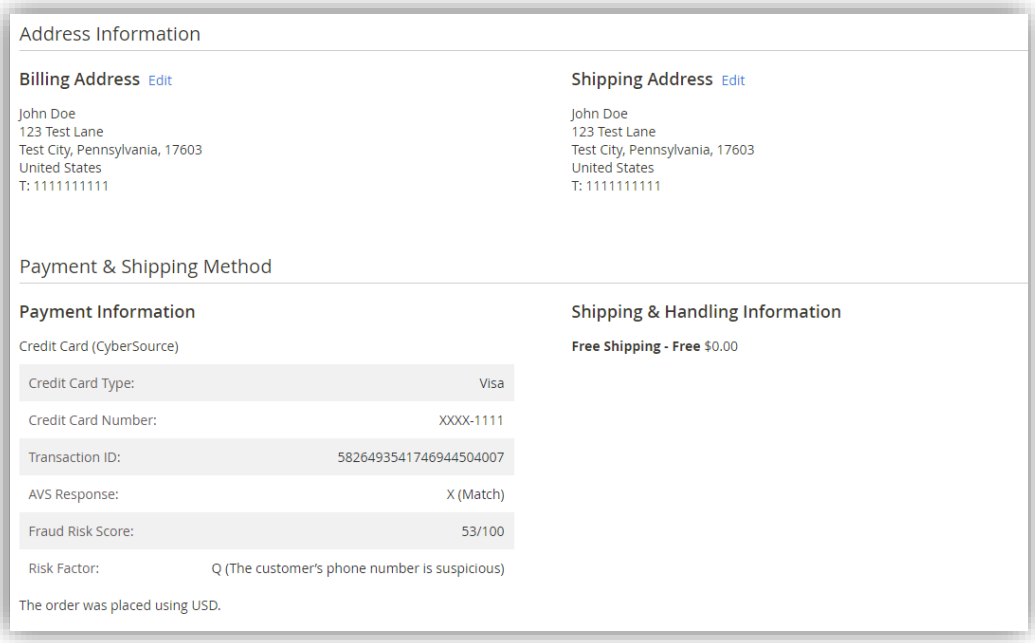
Admin order form

The admin form has the same options and behavior as frontend checkout.



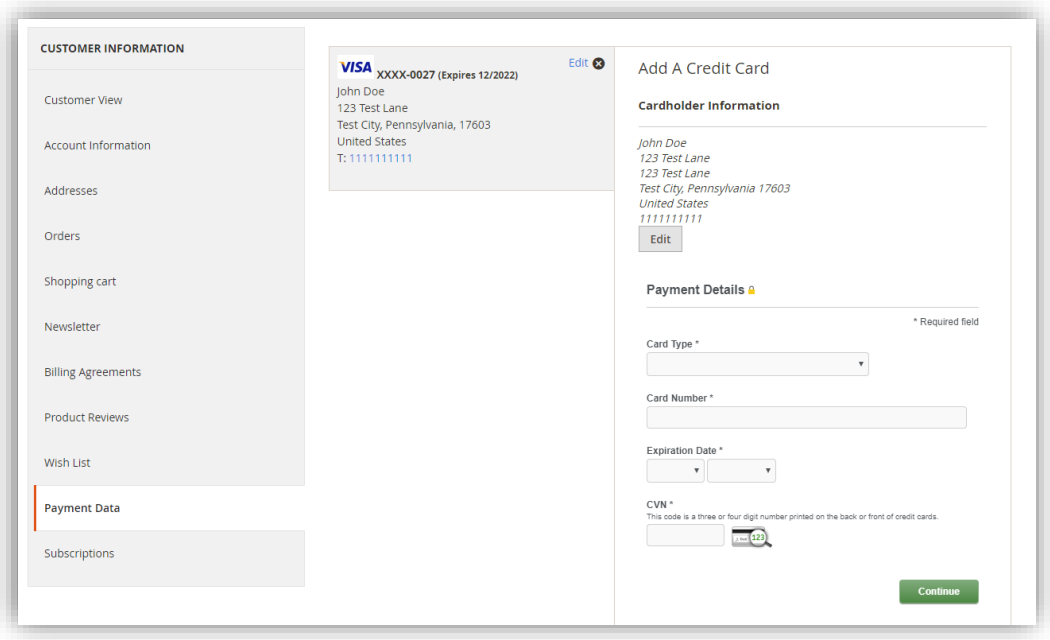
Admin order status page

The admin panel shows extended payment info after placing an order, including transaction ID, validation responses, and fraud score (if using Decision Manager). These details are not visible to the customer at any time.



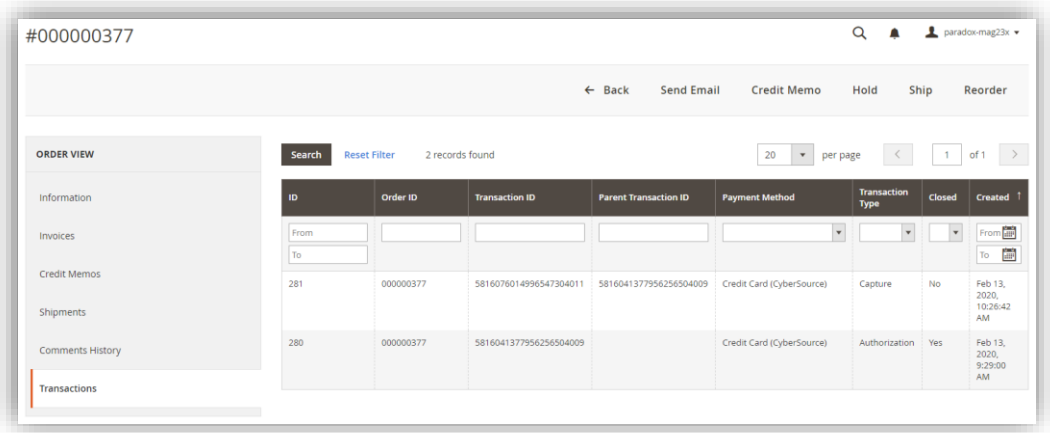
Admin customer ‘Payment Data’ account area

Viewing a customer, you will see an added ‘Payment Data’ tab. This shows all of the same information with all of the same functionality as the equivalent frontend section. You can use this to view, add, edit, and delete your customers’ stored CyberSource payment data.

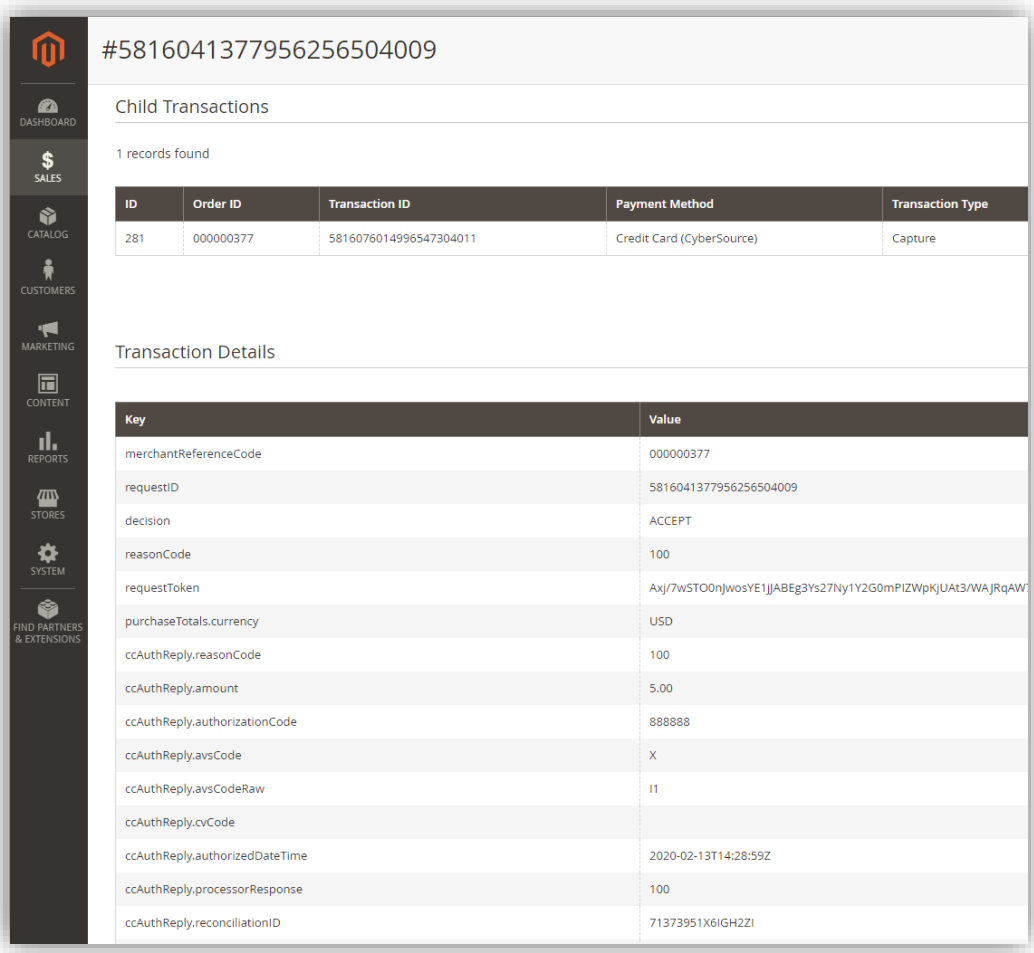


Admin transaction info

Viewing an order, you can also see full transaction info from the ‘Transactions’ tab.



Click into a transaction, and you’ll see all of the raw transaction data returned by the CyberSource Simple Order API. This same data is accessible from the transaction record in code.


DASHBOARD
SALES
CATALOG
CUSTOMERS
MARKETING
CONTENT
REPORTS
STORES
SYSTEM
FIND PARTNERS & EXTENSIONS
#581604137795625650 4009
Child Transactions
1 records found
ID
Order ID
Transaction ID
Payment Method
Transaction Type
281
000000377
581607601499654730 4011
Credit Card (CyberSource)
Capture
Transaction Details
Key
Value
merchantReferenceCode
000000377
requestID
581604137795625650 4009
decision
ACCEPT
reasonCode
100
requestToken
Axj/7wSTO0nJwosYE1jJABEg3Ys27Ny1Y2G0mPIZWpKjUAt3/WAJRqAW
purchaseTotals.currency
USD
ccAuthReply.reasonCode
100
ccAuthReply.amount
5.00
ccAuthReply.authorizationCode
888888
ccAuthReply.avsCode
X
ccAuthReply.avsCodeRaw
I1
ccAuthReply.cvCode
ccAuthReply.authorizedDateTime
2020-02-13T14:28:59Z
ccAuthReply.processorResponse
100
ccAuthReply.reconciliationID
71373951X6IGH2ZI

Frequently Asked Questions & Troubleshooting

Please search our solution directory for the latest answers to common questions and issues:

https://paradoxlabs.freshdesk.com/support/solutions

If your question is not answered there, open a support ticket and we'll help you out.

Is ParadoxLabs CyberSource Payments PCI Compliance?

PCI compliance is a complex and multifaceted issue, covering every aspect of your business. We can't guarantee that your business is PCI-compliant. That depends on your server, passwords, business processes, regular security scans, any other payment methods, and a lot more. What we can tell you is that this extension will not prevent you from being PCI compliant. We don't log confidential cardholder data or do anything else that would bring you under scrutiny.

This extension implements **Secure Acceptance Hosted Checkout** iframes for all credit card forms, and does not support collecting credit card data by any other means. That makes the ParadoxLabs CyberSource payment method eligible for **PCI v3.2 Self-Assessment Questionnaire A (SAQ A)**, the simplest possible SAQ form.

Note that you **must** have SSL (TLS) enabled on all checkout and login forms, and that this eligibility **only** applies to this specific payment method. Any other payment methods or credit card handling your business may perform will have its own SAQ eligibility, and may require you to complete a more stringent SAQ form (A-EP or D).

For details on the SAQ types and what eligibility means, see "Self-Assessment Questionnaire Instructions and Guidelines (3.2)" (PDF, by PCI Standards Security Council).

How do I do an online refund from Magento?

In order to process an 'online' refund through CyberSource, you have to go to the **invoice** you want to refund, and click the 'Credit Memo' button from there.

If you've done that correctly, at the bottom of the page you should see a button that says 'Refund'.

If you only have one button that says 'Refund Offline', it's because you clicked 'Credit Memo' from the order instead of from the invoice.

The reason for this is that the refund needs to be associated with a particular capture transaction. An order can contain any number of capture transactions, but every capture has an invoice that's directly related. You refund an invoice, not an order.

How does this payment method handle currency?

Transactions are processed in the base currency for the website customers are purchasing from. Any alternate currencies selected on the frontend are converted to the website's base currency by Magento's built-in currency handling, based on conversion rates provided by a web service configured in your Magento Admin Panel.

Magento allows for a separate base currency per website, if configured to do so. In order to define explicit prices in multiple currencies, each currency must have its own website where it is set as the base currency. All currency setup is configured outside of our payment extension settings.

Your CyberSource account must allow transactions to be processed in your website's base currency(s).

Technical / Integration Details

Architecture

The payment method code for this method is `paradoxlabs_cybersource`.

`ParadoxLabs_CyberSource` is the payment method module, built heavily on the `ParadoxLabs_TokenBase` module. TokenBase defines a variety of interfaces and architecture for handling tokenization and stored cards cleanly.

The payment method class is `\ParadoxLabs\CyberSource\Model\Method`. This talks to CyberSource primarily through `\ParadoxLabs\CyberSource\Model\Gateway` (via PHP's SoapClient and classes in `\ParadoxLabs\CyberSource\Gateway\Api`), and stores card metadata in instances of `\ParadoxLabs\CyberSource\Model\Card`. Each of these extends an equivalent abstract class in TokenBase, and implements only the details specific to the CyberSource API.

Card instances are stored in table `paradoxlabs_stored_card`, and referenced by quotes and orders via a `tokenbase_id` column on tables `quote_payment` and `sales_order_payment`.

In all cases, we strongly discourage any customization by editing our code directly. We cannot support customizations. Use Magento's preferences or plugins to modify behavior if necessary. If your use case isn't covered, let us know.

Custom database schema

- Added table: `paradoxlabs_stored_card`
- Added column: `quote_payment.tokenbase_id`
- Added column: `sales_order_payment.tokenbase_id`

Events

- `tokenbase_before_load_payment_info` (`method`, `customer`, `transport`, `info`): Fires before preparing method-specific information for the order payment info blocks (frontend, admin, and emails). Use this to add additional information to the payment info block.
- `tokenbase_after_load_payment_info` (`method`, `customer`, `transport`, `info`): Fires before preparing method-specific information for the order payment info blocks (frontend, admin, and emails). Use this to add additional information to the payment info block, or modify what's there by default.
- `tokenbase_before_load_active_cards` (`method`, `customer`): Fires before loading a customer's available stored cards.
- `tokenbase_after_load_active_cards` (`method`, `customer`, `cards`): Fires after loading a customer's available stored cards. Use this to modify cards available to the customer or admin.

Magento API: REST and SOAP

This module supports the Magento API via standard interfaces. You can use it to create, read, update, and delete stored cards.

If you have a specific use case in mind that is not covered, please let us know.

You can generate new cards by creating a new TokenBase Card with a valid CyberSource credit card token (and associated card and address data). To place an order with a stored card, pass that card's hash in as `additional_data -> card_id`, along with `cc_cid` for the CCV (if any), `save` (true or false) for whether it should be saved or not, and `response_jwt` if 3D Secure is involved.

Note that raw credit card numbers (`cc_number`) will not be accepted. You must store the card in CyberSource TMS via one of CyberSource's various APIs, then store the resulting TMS token as a card, prior to using it for an order. See CyberSource documentation on how to tokenize a credit card in various circumstances. We describe the process below for doing so with the CyberSource Secure Acceptance iframe integration method: '*How-To: API Checkout Flow*'.

To support Payer Authentication outside Magento, you must use the `cardinal` fields in `checkoutConfig` to implement the Cardinal Cruise API JS Hybrid integration, then pass the resulting JSON Web Token (JWT) with the rest of the order payment data as `response_jwt`. The process is rather involved, but detailed below in '*How-To: API 3D Secure Flow*'.

The extension's custom REST API requests are detailed below. Some response data has been omitted for brevity.

Create and update (POST, PUT) requests take three objects: `card` with primary card data, `address` with address information, and `additional` for card metadata. In responses, `address` and `additional` will be nested within `card` as `address_object` and `additional_object`. This is done for technical reasons. The data formats differ, and not all fields that are returned can be set via API (EG `in_use`, `label`). This means you cannot take a card record and directly post it back to the API to update.

Integration / Admin-Authenticated API Endpoints

These API requests allow solutions acting with an admin user login, OAUTH authentication, or token-based authentication to take action on any card in the system. Data and behavior are not limited.

GET /V1/tokenbase/:cardId (get one card by ID)

Example request:

```
GET /rest/V1/tokenbase/1 HTTP/1.1
Host: {host}
Authorization: Bearer {api_key}
```

Example response:

```
{
    "id": 1,
    "in_use": true,
    "additional_object": {
        "cc_type": "VI",
        "cc_last4": "0027",
        "cc_exp_year": "2022",
        "cc_exp_month": "12",
        "cc_bin": "400700",
        "fingerprint": "7010000000008030027"
    },
```

```
    "address_object": {
        "region": {
            "region_code": "PA",
            "region": "Pennsylvania",
            "region_id": 51
        },
        "region_id": 51,
        "country_id": "US",
        "street": [
            "123 Test Ln."
        ],
        "company": "",
        "telephone": "111-111-1111",
        "postcode": "17603",
        "city": "Lancaster",
        "firstname": "John",
        "lastname": "Doe"
    },
    "customer_email": "email@example.com",
    "customer_id": 1,
    "customer_ip": "127.0.0.1",
    "profile_id": null,
    "payment_id": "9F6B429B2C2A64D5E05341588E0A3F70",
    "method": "paradoxlabs_cybersource",
    "hash": "f7d085165acdfa0ea6a0b...770111",
    "active": "1",
    "created_at": "2017-08-03 16:31:54",
    "updated_at": "2017-09-20 14:24:14",
    "last_use": "2017-08-03 16:31:54",
    "expires": "2019-06-30 23:59:59",
    "label": "Visa XXXX-0027"
}
```


GET /V1/tokenbase/search (get multiple cards, with searchCriteria)

Example request:

```
GET /rest/V1/tokenbase/search?searchCriteria[pageSize]=1 HTTP/1.1
Host: {host}
Authorization: Bearer {api_key}
```

Example response:

```
{
    "items": [
        {
            "id": 1,
            // ... other card info
        }
    ],
    "search_criteria": {
        "filter_groups": [],
        "page_size": 1
    },
    "total_count": 51
}
```

See also: Search using REST APIs (Magento DevDocs)

POST /V1/tokenbase (create card)

Example request:

```
POST /rest/V1/tokenbase HTTP/1.1
Host: {host}
Authorization: Bearer {api_key}
Content-Type: application/json

{
    "card": {
        "customer_email": "email@example.com",
        "customer_id": 1,
        "customer_ip": "",
        "profile_id": null,
        "payment_id": "9F6B429B2C2A64D5E05341588E0A3F70",
```

```
        "method": "paradoxlabs_cybersource",
        "active": "1",
        "created_at": "2017-08-03 16:31:54",
        "last_use": "2017-08-03 16:31:54",
        "expires": "2019-06-30 23:59:59"
    },
    "address": {
        "region": {
            "region_code": "PA",
            "region": "Pennsylvania",
            "region_id": 51
        },
        "region_id": 51,
        "country_id": "US",
        "street": [
            "123 Test Ln."
        ],
        "company": "",
        "telephone": "111-111-1111",
        "postcode": "17603",
        "city": "Lancaster",
        "firstname": "John",
        "lastname": "Doe",
        "vat_id": ""
    },
    "additional": {
        "cc_exp_month": "01",
        "cc_exp_year": "2023",
        "cc_last4": "0027",
        "cc_type": "VI",
        "cc_bin": "400700",
        "fingerprint": "70100000000008030027"
    }
}
```


Example response:


```
{
    "id": 95,
    "in_use": false,
    "additional_object": {
        "cc_type": "VI",
        "cc_last4": "0027",
        "cc_exp_year": "2023",
        "cc_exp_month": "01",
        "cc_country": "US",
        "cc_bin": "400700",
        "fingerprint": "70100000000008030027"
    },
    "address_object": {
        "region": {
            "region_code": "PA",
            "region": "Pennsylvania",
            "region_id": 51
        },
        "region_id": 51,
        "country_id": "US",
        "street": [
            "123 Test Ln."
        ],
        "company": "",
        "telephone": "111-111-1111",
        "postcode": "17603",
        "city": "Lancaster",
        "firstname": "John",
        "lastname": "Doe",
    },
    "customer_email": "email@example.com",
    "customer_id": 1,
    "customer_ip": "127.0.0.1",
    "profile_id": null,
    "payment_id": "9F6B429B2C2A64D5E05341588E0A3F70",
    "method": "paradoxlabs_cybersource",
    "hash": "9b83d4683f3d3...2309ccd65b",
    "active": "1",
    "created_at": "2017-09-25 17:41:21",
    "updated_at": "2017-09-25 17:41:21",
    "last_use": "2017-08-03 16:31:54",
```

```
    "expires": "2023-01-31 23:59:59",
    "label": "Visa XXXX-0027"
}
```

PUT /V1/tokenbase/:cardId (update card)

Example request:

```
PUT /rest/V1/tokenbase/1 HTTP/1.1
Host: {host}
Authorization: Bearer {api_key}
Content-Type: application/json

{
    "card": {
        "id": 1,
        "customer_email": "email@example.com",
        "customer_id": 1,
        "customer_ip": "127.0.0.1",
        "profile_id": null,
        "payment_id": "9F6B429B2C2A64D5E05341588E0A3F70",
        "method": "paradoxlabs_cybersource",
        "hash": "f7d085165acdfa0ea6a0b...770111",
        "active": "1",
        "created_at": "2017-08-03 16:31:54",
        "last_use": "2017-08-03 16:31:54",
        "expires": "2019-06-30 23:59:59"
    },
    "address": {
        "region": {
            "region_code": "PA",
            "region": "Pennsylvania",
            "region_id": 51
        },
        "region_id": 51,
        "country_id": "US",
        "street": [
            "123 Test Ln."
        ],
        "company": "",
        "telephone": "111-111-1111",
        "postcode": "17603",
        "city": "Lancaster",
        "firstname": "John",
        "lastname": "Doe",
        "vat_id": ""
    },
    "additional": {
        "cc_exp_month": "01",
        "cc_exp_year": "2023",
        "cc_last4": "0027",
        "cc_type": "VI",
        "cc_bin": "400700",
        "fingerprint": "70100000000008030027"
    }
}
```

Example response:

```
{
    "id": 1,
    "in_use": false,
    "additional_object": {
        "cc_type": "VI",
        "cc_last4": "0027",
        "cc_exp_year": "2023",
        "cc_exp_month": "01",
        "cc_bin": "400700",
        "fingerprint": "70100000000008030027"
    },
    "address_object": {
        "region": {
            "region_code": "PA",
            "region": "Pennsylvania",
```

```
            "region_id": 51
        },
        "region_id": 51,
        "country_id": "US",
        "street": [
            "123 Test Ln."
        ],
        "company": "",
        "telephone": "111-111-1111",
        "postcode": "17603",
        "city": "Lancaster",
        "firstname": "John",
        "lastname": "Doe",
    },
    "customer_email": "email@example.com",
    "customer_id": 1,
    "customer_ip": "127.0.0.1",
    "profile_id": null,
    "payment_id": "9F6B429B2C2A64D5E05341588E0A3F70",
    "method": "paradoxlabs_cybersource",
    "hash": " f7d085165acdfa0ea6a0b...770111",
    "active": "1",
    "created_at": "2017-09-25 17:41:21",
    "updated_at": "2017-09-25 17:41:21",
    "last_use": "2017-08-03 16:31:54",
    "expires": "2023-01-31 23:59:59",
    "label": "Visa XXXX-0027"
}
```


DELETE /V1/tokenbase/:cardId (delete card by ID)
Example request:

```
DELETE /rest/V1/tokenbase/95 HTTP/1.1
Host: {host}
Authorization: Bearer {api_key}
```

Example response:

```
True
```

Customer Authenticated API Endpoints

These API requests allow authenticated frontend customers to manage their stored cards. This is intended for headless implementations or app integration where card management needs to be exposed outside of Magento's standard frontend.

Customers will only be able to access and manipulate active cards assigned to their specific customer ID.

Note: These requests are disabled by default. You can enable them at **Admin > Stores > Configuration > Sales > Checkout > ParadoxLabs Payment Module Settings > Enable public API**. Only enable this if you use them.

GET /V1/tokenbase/mine/:cardHash (get one card by hash)
Example request:

```
GET /rest/V1/tokenbase/mine/50b8e326b012e793957215c0361afc4b52434b26 HTTP/1.1
Host: {host}
Authorization: Bearer {api_key}
```

Example response:


```
{
    "id": 1,
    "in_use": true,
    "additional_object": {
        "cc_type": "VI",
        "cc_last4": "0027",
```

```
        "cc_exp_year": “2023”,
        "cc_exp_month": "01",
        “cc_bin”: “400700”,
        "fingerprint": "70100000000008030027"
    },
    "address_object": {
        "region": {
            "region_code": "PA",
            "region": "Pennsylvania",
            "region_id": 51
        },
        "region_id": 51,
        "country_id": "US",
        "street": [
            "123 Test Ln."
        ],
        "company": "",
        "telephone": "111-111-1111",
        "postcode": "17603",
        "city": "Lancaster",
        "firstname": "John",
        "lastname": "Doe"
    },
    "customer_email": "email@example.com",
    "customer_id": 1,
    "customer_ip": "127.0.0.1",
    "profile_id": null,
    "payment_id": "9F6B429B2C2A64D5E05341588E0A3F70",
    "method": "paradoxlabs_cybersource",
    "hash": "50b8e326b012e793957215c0361afc4b52434b26",
    "active": "1",
    "created_at": "2017-08-03 16:31:54",
    "updated_at": "2017-09-20 14:24:14",
    "last_use": "2017-08-03 16:31:54",
    "expires": "2023-01-31 23:59:59",
    "label": "Visa XXXX-0027"
}
```

GET /V1/tokenbase/mine/search (get multiple cards, with searchCriteria)

Example request:

```
GET /rest/V1/tokenbase/mine/search?searchCriteria[pageSize]=3 HTTP/1.1
Host: {host}
Authorization: Bearer {api_key}
```

Example response:

```
{
    "items": [
        {
            "id": 1,
            // ... other card info
        }
    ],
    "search_criteria": {
        "filter_groups": [],
        "page_size": 3
    },
    "total_count": 5
}
```

See also: Search using REST APIs (Magento DevDocs)

POST /V1/tokenbase/mine (create card)

Example request:

```
POST /rest/V1/tokenbase/mine HTTP/1.1
Host: {host}
Content-Type: application/json
Authorization: Bearer {api_key}

{
    "card": {
```

```
        "customer_email": "email@example.com",
        "customer_id": 1,
        "customer_ip": "127.0.0.1",
        "profile_id": null,
        "payment_id": "9F6B429B2C2A64D5E05341588E0A3F70",
        "method": "paradoxlabs_cybersource",
        "active": "1"
    },
    "address": {
        "region": {
            "region_code": "PA",
            "region": "Pennsylvania",
            "region_id": 51
        },
        "region_id": 51,
        "country_id": "US",
        "street": [
            "123 Test Ln."
        ],
        "company": "",
        "telephone": "111-111-1111",
        "postcode": "17603",
        "city": "Lancaster",
        "firstname": "John",
        "lastname": "Doe",
        "vat_id": ""
    },
    "additional": {
        "cc_type": “VI”,
        "cc_last4": “0027”,
        "cc_exp_year": “2023”,
        "cc_exp_month": "01",
        “cc_bin”: “400700”,
        "fingerprint": "70100000000008030027"
    }
}
```


Example response:


```
{
    "id": 95,
    "in_use": false,
    "additional_object": {
        "cc_type": “VI”,
        "cc_last4": “0027”,
        "cc_exp_year": “2023”,
        "cc_exp_month": "01",
        “cc_bin”: “400700”,
        "fingerprint": "70100000000008030027"
    },
    "address_object": {
        "region": {
            "region_code": "PA",
            "region": "Pennsylvania",
            "region_id": 51
        },
        "region_id": 51,
        "country_id": "US",
        "street": [
            "123 Test Ln."
        ],
        "company": "",
        "telephone": "111-111-1111",
        "postcode": "17603",
        "city": "Lancaster",
        "firstname": "John",
        "lastname": "Doe",
    },
    "customer_email": "email@example.com",
    "customer_id": 1,
    "customer_ip": "127.0.0.1",
    "profile_id": null,
    "payment_id": "9F6B429B2C2A64D5E05341588E0A3F70",
    "method": "paradoxlabs_cybersource",
    "hash": "9b83d4683f3d3...2309ccd65b",
    "active": "1",
    "created_at": "2017-09-25 17:41:21",
    "updated_at": "2017-09-25 17:41:21",
```

```
    "last_use": "2017-08-03 16:31:54",
    "expires": "2023-01-31 23:59:59",
    "label": "Visa XXXX-0027"
}
```

PUT /V1/tokenbase/mine/:cardHash (update card by hash)

Example request:

```
PUT /rest/V1/tokenbase/mine/50b8e326b012e793957215c0361afc4b52434b26 HTTP/1.1
Host: {host}
Authorization: Bearer {api_key}
Content-Type: application/json

{
    "card": {
        "id": 1,
        "customer_email": "email@example.com",
        "customer_id": 1,
        "customer_ip": "127.0.0.1",
        "profile_id": null,
        "payment_id": "9F6B429B2C2A64D5E05341588E0A3F70",
        "method": "paradoxlabs_cybersource",
        "hash": "50b8e326b012e793957215c0361afc4b52434b26",
        "active": "1",
        "expires": "2023-01-31 23:59:59"
    },
    "address": {
        "region": {
            "region_code": "PA",
            "region": "Pennsylvania",
            "region_id": 51
        },
        "region_id": 51,
        "country_id": "US",
        "street": [
            "123 Test Ln."
        ],
        "company": "",
        "telephone": "111-111-1111",
        "postcode": "17603",
        "city": "Lancaster",
        "firstname": "John",
        "lastname": "Doe",
        "vat_id": ""
    },
    "additional": {
        "cc_type": “VI”,
        "cc_last4": “0027”,
        "cc_exp_year": “2023”,
        "cc_exp_month": "01",
        “cc_bin”: “400700”,
        "fingerprint": "70100000000008030027"
    }
}
```

Example response:

```
{
    "id": 1,
    "in_use": false,
    "additional_object": {
        "cc_type": “VI”,
        "cc_last4": “0027”,
        "cc_exp_year": “2023”,
        "cc_exp_month": "01",
        “cc_bin”: “400700”,
        "fingerprint": "70100000000008030027"
    },
    "address_object": {
        "region": {
            "region_code": "PA",
            "region": "Pennsylvania",
            "region_id": 51
        },
        "region_id": 51,
        "country_id": "US",
```

```
        "street": [
            "123 Test Ln."
        ],
        "company": "",
        "telephone": "111-111-1111",
        "postcode": "17603",
        "city": "Lancaster",
        "firstname": "John",
        "lastname": "Doe",
    },
    "customer_email": "email@example.com",
    "customer_id": 1,
    "customer_ip": "127.0.0.1",
    "profile_id": null,
    "payment_id": "9F6B429B2C2A64D5E05341588E0A3F70",
    "method": "paradoxlabs_cybersource",
    "hash": "50b8e326b012e793957215c0361afc4b52434b26",
    "active": "1",
    "created_at": "2017-09-25 17:41:21",
    "updated_at": "2017-09-25 17:41:21",
    "last_use": "2017-08-03 16:31:54",
    "expires": "2023-01-31 23:59:59",
    "label": "Visa XXXX-0027"
}
```


DELETE /V1/tokenbase/mine/:cardHash (delete card by hash)

Example request:

```
DELETE /rest/V1/tokenbase/mine/50b8e326b012e793957215c0361afc4b52434b26 HTTP/1.1
Host: {host}
Authorization: Bearer {api_key}
```

Example response:

```
True
```

Guest API Endpoints

These API requests allow unauthenticated frontend guest users to add and fetch an individual stored card. This is intended for headless implementations or app integration where card management needs to be exposed outside of Magento's standard frontend. Guests are not able to list, edit, delete, or reuse stored cards, so no API requests are exposed for those actions.

Guests will only be able to access and manipulate active cards, by hash, not assigned to any customer ID.

Note: These requests are disabled by default. You can enable them at **Admin > Stores > Configuration > Sales > Checkout > ParadoxLabs Payment Module Settings > Enable public API**. Only enable this if you use them.

GET /V1/tokenbase/guest/:cardHash (get one card by hash)

Example request:

```
GET /rest/V1/tokenbase/guest/50b8e326b012e793957215c0361afc4b52434b26 HTTP/1.1
Host: {host}
```

Example response:


```
{
    "id": 1,
    "in_use": true,
    "additional_object": {
        "cc_type": "VI",
        "cc_last4": "0027",
        "cc_exp_year": "2023",
        "cc_exp_month": "01",
        "cc_bin": "400700",
        "fingerprint": "70100000000008030027"
```

```
    },
    "address_object": {
        "region": {
            "region_code": "PA",
            "region": "Pennsylvania",
            "region_id": 51
        },
        "region_id": 51,
        "country_id": "US",
        "street": [
            "123 Test Ln."
        ],
        "company": "",
        "telephone": "111-111-1111",
        "postcode": "17603",
        "city": "Lancaster",
        "firstname": "John",
        "lastname": "Doe"
    },
    "customer_email": "email@example.com",
    "customer_id": 0,
    "customer_ip": "127.0.0.1",
    "profile_id": null,
    "payment_id": "9F6B429B2C2A64D5E05341588E0A3F70",
    "method": "paradoxlabs_cybersource",
    "hash": "50b8e326b012e793957215c0361afc4b52434b26",
    "active": "1",
    "created_at": "2017-08-03 16:31:54",
    "updated_at": "2017-09-20 14:24:14",
    "last_use": "2017-08-03 16:31:54",
    "expires": "2023-01-31 23:59:59",
    "label": "Visa XXXX-0027"
}
```

POST /V1/tokenbase/guest (create card)

Example request:

```
POST /rest/V1/tokenbase/guest HTTP/1.1
Host: {host}
Content-Type: application/json

{
    "card": {
        "customer_email": "email@example.com",
        "customer_id": 0,
        "customer_ip": "127.0.0.1",
        "profile_id": null,
        "payment_id": "9F6B429B2C2A64D5E05341588E0A3F70",
        "method": "paradoxlabs_cybersource",
        "active": "1"
    },
    "address": {
        "region": {
            "region_code": "PA",
            "region": "Pennsylvania",
            "region_id": 51
        },
        "region_id": 51,
        "country_id": "US",
        "street": [
            "123 Test Ln."
        ],
        "company": "",
        "telephone": "111-111-1111",
        "postcode": "17603",
        "city": "Lancaster",
        "firstname": "John",
        "lastname": "Doe",
        "vat_id": ""
    },
    "additional": {
        "cc_type": “VI”,
        "cc_last4": “0027”,
        "cc_exp_year": “2023”,
        "cc_exp_month": "01",
        “cc_bin”: “400700”,
        "fingerprint": "7010000000008030027"
    }
```

```
}
```

Example response:


```
{
    "id": 95,
    "in_use": false,
    "additional_object": {
        "cc_type": "VI",
        "cc_last4": "0027",
        "cc_exp_year": "2023",
        "cc_exp_month": "01",
        "cc_bin": "400700",
        "fingerprint": "70100000000008030027"
    },
    "address_object": {
        "region": {
            "region_code": "PA",
            "region": "Pennsylvania",
            "region_id": 51
        },
        "region_id": 51,
        "country_id": "US",
        "street": [
            "123 Test Ln."
        ],
        "company": "",
        "telephone": "111-111-1111",
        "postcode": "17603",
        "city": "Lancaster",
        "firstname": "John",
        "lastname": "Doe",
    },
    "customer_email": "email@example.com",
    "customer_id": 0,
    "customer_ip": "127.0.0.1",
    "profile_id": null,
    "payment_id": "9F6B429B2C2A64D5E05341588E0A3F70",
    "method": "paradoxlabs_cybersource",
    "hash": "9b83d4683f3d3...2309ccd65b",
    "active": "1",
    "created_at": "2017-09-25 17:41:21",
    "updated_at": "2017-09-25 17:41:21",
    "last_use": "2017-08-03 16:31:54",
    "expires": "2023-01-31 23:59:59",
    "label": "Visa XXXX-0027"
}
```


Magento API: GraphQL

For Magento 2.3.1+, this extension supports the GraphQL API for all customer card management. This is intended for PWA and headless implementations where card management needs to be exposed outside of Magento's standard frontend.

Customers will only be able to access and manipulate active cards assigned to their specific customer ID.

Guests will only be able to access and manipulate active cards, by hash, not assigned to any customer ID.

Note: These requests are disabled by default. You can enable them at **Admin > Stores > Configuration > Sales > Checkout > ParadoxLabs Payment Module Settings > Enable public API**. Only enable this if you use them.

We recommend using the Chrome Altair GraphQL Client browser extension for browsing your store's GraphQL schema and testing API requests.

Note that raw credit card numbers (`cc_number`) will not be accepted. You must store the card in CyberSource TMS via one of CyberSource's various APIs, then store the resulting TMS token as a card, prior to using it for an order. See CyberSource documentation on how to tokenize a credit card in various circumstances. We describe the

process below for doing so with the CyberSource Secure Acceptance iframe integration method: '*How-To: API Checkout Flow*'.

To support Payer Authentication outside Magento, you must use the `cardinal` fields in `checkoutConfig` to implement the Cardinal Cruise API JS Hybrid integration, then pass the resulting JSON Web Token (JWT) with the rest of the order payment data as `response_jwt`. The process is rather involved, but detailed below in '*How-To: API 3D Secure Flow*'.

Queries

tokenBaseCards(hash: String): [TokenBaseCard]
Get the current customer's stored card(s), if any. Takes a card hash (optional); returns one or more `TokenBaseCard` records. If no hash is given, will return all active cards belonging to the customer.

tokenBaseCheckoutConfig(method: String!): TokenBaseCheckoutConfig
Get checkout configuration for the given TokenBase payment method. Takes a TokenBase payment method code, such as `paradoxlabs_cybersource`; returns a `TokenBaseCheckoutConfig`. This returns all data necessary to render and handle the client-side checkout form. Values mirror what is passed to Magento's standard frontend checkout.

cyberSourceSecureAcceptParams(input: TokenBaseCyberSourceSecureAcceptInput!): TokenBaseCyberSourceSecureAcceptParams
Get the CyberSource Secure Acceptance form URL and parameters for collecting payment data. Takes a source key (checkout or paymentinfo), and a cart ID (for checkout) or card ID (for card updates), as well as guestEmail and billingAddress. Information will be pulled from the checkout quote when possible. Response data is in the form of a URL (for the iframe src), and parameters to be posted to that iframe, in order to load the Secure Acceptance payment form.

cyberSourceCardinalCruiseAuthPayload(input: TokenBaseCyberSourceCardinalCruiseAuthInput!): TokenBaseCyberSourceCardinalCruiseParams
Get the CyberSource Cardinal Cruise parameters for completing payer authentication. If the customer experiences a 3D Secure error at checkout ('payer authentication required', response code 475), query the Cardinal Cruise params using the cart ID. This will give you the encoded data necessary to process the authentication with the Cardinal Cruise Hybrid JS API. See the 'how-to' guide on 3D Secure flow later.

Mutations

createTokenBaseCard(input: TokenBaseCardCreateInput!): TokenBaseCard
Create a new stored card. Takes `TokenBaseCardCreateInput`, returns the new stored `TokenBaseCard` if successful.

deleteTokenBaseCard(hash: String!): Boolean
Delete a stored card. Takes a card hash; returns true if successful.

updateTokenBaseCard(input: TokenBaseCardUpdateInput!): TokenBaseCard
Update an existing stored card. Takes `TokenBaseCardUpdateInput`; returns the updated `TokenBaseCard` if successful.

Data Types

TokenBaseCard
A stored payment account/credit card.

```
type TokenBaseCard {
        hash: String                                    Card identifier hash
        address: CustomerAddress                        Card billing address
```

```
        customer_email: String                          Customer email
        customer_id: Int                                Customer ID
        customer_ip: String                             Created-by IP
        profile_id: String                              Card gateway profile ID
        payment_id: String                              Card gateway payment ID
        method: String                                  Payment method code
        active: Boolean                                 Is card active
        created_at: String                              Created-at date
        updated_at: String                              Last updated date
        last_use: String                                Last used date
        expires: String                                 Expiration date
        label: String                                   Card label
        additional: TokenBaseCardAdditional             Card payment data
}
```

TokenBaseCardAdditional

Details and metadata for a stored CC/ACH.

```
type TokenBaseCardAdditional {
        cc_type: String                                 CC Type
        cc_owner: String                                CC Owner
        cc_bin: String                                  CC Bin (CC First-6)
        cc_last4: String                                CC Last-4
        cc_exp_year: String                             CC Expiration Year
        cc_exp_month: String                            CC Expiration Month
        echeck_account_name: String                     ACH Account Name
        echeck_bank_name: String                        ACH Bank Name
        echeck_account_type: TokenBaseEcheckAccountType ACH Account type
        echeck_routing_number_last4: String             ACH Routing Number Last-4
        echeck_account_number_last4: String             ACH Account Number Last-4
}
```

TokenBaseCheckoutConfig

Checkout configuration for a TokenBase payment method.

```
type TokenBaseCheckoutConfig {
        method: String                                  Payment method code
        useVault: Boolean                               Are stored cards enabled?
        canSaveCard: Boolean                            Can cards be saved?
        forceSaveCard: Boolean                          Is card saving forced?
        defaultSaveCard: Boolean                        Hash of the default card to select
        isCcDetectionEnabled: Boolean                   Is CC type detection enabled?
        logoImage: String                               Payment logo image URL (if enabled)
        requireCcv: Boolean                             Is CVV required for stored cards?
        sandbox: Boolean                                Is the payment gateway in sandbox mode?
        canStoreBin: Boolean                            Is CC BIN (CC first6) storage enabled?
        availableTypes: [TokenBaseKeyValue]             Available CC types
        months: [TokenBaseKeyValue]                     Available CC Exp Months
        years: [TokenBaseKeyValue]                      Available CC Exp Years
        hasVerification: Boolean                        Is CVV enabled?
        cvvImageUrl: String                             CVV helper image URL
        paramUrl: String                                URL for retrieving Secure Acceptance request params
        fingerprintUrl: String                          Script URL for fingerprinting, if enabled
        cardinalScript: String                          Script URL for Cardinal Cruise Songbird.js
        cardinalAuthUrl: String                         URL for retrieving Cardinal Cruise enroll params
        cardinalJWT: String                             JWT payload for initializing Cardinal Cruise API
}
```

TokenBaseKeyValue

Container for generic key/value data.

```
type TokenBaseKeyValue {
        key: String                                     Generic key
        value: String                                   Generic value
}
```

TokenBaseCardUpdateInput

Input for updating a stored card.

```
input TokenBaseCardUpdateInput {
        hash: String!                                   Card identifier hash to update (required)
```

```
        address: CustomerAddressInput                   Card billing address
        customer_email: String                          Customer email
        customer_ip: String                             Created-by IP
        method: String                                  Payment method code
        active: Boolean                                 Is card active
        expires: String                                 Card expiration date (YYYY-MM-DD 23:59:59)
        additional: TokenBaseCardPaymentInput           Card payment data
}
```

TokenBaseCardCreateInput

Input for creating a stored card.

```
input TokenBaseCardCreateInput {
        address: CustomerAddressInput                   Card billing address
        customer_email: String!                         Customer email (required)
        customer_ip: String                             Created-by IP
        method: String!                                 Payment method code (required)
        active: Boolean                                 Is card active
        expires: String                                 Card expiration date (YYYY-MM-DD 23:59:59)
        additional: TokenBaseCardPaymentInput           Card payment data
}
```

TokenBaseCardPaymentInput

Payment data for a stored card. Note, the specific fields that are relevant depend on the payment method. This data structure is also used for adding payment data to the cart during checkout.

```
input TokenBaseCardPaymentInput {
        cc_type: String                                 CC Type
        cc_owner: String                                CC Owner
        cc_bin: String                                  CC Bin (CC First-6)
        cc_last4: String                                CC Last-4
        cc_number: String                               CC Number
        cc_cid: String                                  CC CVV
        cc_exp_year: String                             CC Expiration Year
        cc_exp_month: String                            CC Expiration Month
        echeck_account_name: String                     ACH Account Name
        echeck_bank_name: String                        ACH Bank Name
        echeck_account_type: TokenBaseEcheckAccountType ACH Account Type
        echeck_routing_number: String                   ACH Routing Number
        echeck_account_number: String                   ACH Account Number
        token: String                                   Stripe Elements card token
        save: Boolean                                   Save the card for later use? (checkout only)
        card_id: String                                 Card identifier hash to use (checkout only)
        response_jwt: String                            Payer Authentication response JWT (checkout only)
}
```

TokenBaseCyberSourceCardinalCruiseAuthInput

Input for obtaining the Cardinal Cruise (3D Secure/Payer Authentication) parameters after a 3DS rejected checkout attempt. These are necessary for completing the payer authentication process.

```
input TokenBaseCyberSourceCardinalCruiseAuthInput {
        cartId: String!                                 Cart/quote ID hash
        cardId: String                                  Card ID hash
        guestEmail: String                              User email address
}
```

TokenBaseCyberSourceCardinalCruiseParams

Output payloads for the Cardinal Cruise Hybrid process. Each value corresponds to a `continue` parameter in the Cardinal Cruise JS API. The authPayload and orderPayload parameters are JSON-encoded and must be decoded before use.

```
type TokenBaseCyberSourceCardinalCruiseParams {
        authPayload: String                             continue() param 2 (JSON)
        orderPayload: String                            continue() param 3 (JSON)
        JWT: String                                     continue() param 4
}
```

TokenBaseCyberSourceSecureAcceptInput

Input for fetching a Secure Acceptance form URL and parameters. The form location affects whether the form is tied to the cart session data (billing address, etc.) or not.

```
input TokenBaseCyberSourceSecureAcceptInput {
        cartId: String                                      Cart/quote ID hash (checkout only)
        cardId: String                                      Card ID hash (paymentinfo only)
        source: TokenBaseCyberSourceSecureAcceptSource!     Form location (checkout or paymentinfo)
        guestEmail: String                                  User email address
        billingAddress: CustomerAddressInput                User billing address
}
```

TokenBaseCyberSourceSecureAcceptParams

Data necessary to load a Secure Acceptance payment form. This consists of a URL, and JSON-encoded parameters. The parameters must be decoded and then POSTed to the iframeAction URL in order to load the form successfully.

```
type TokenBaseCyberSourceSecureAcceptParams {
        iframeAction: String                                URL for the Secure Acceptance payment form iframe
        iframeParams: String                                Parameters for the Secure Acceptance form (JSON)
}
```

GraphQL Query Examples

Some response data has been omitted for brevity.

Fetch card by ID

Example request:

```
{
  tokenBaseCards(hash:"ec431a3e1f9904a35dc083a257cf2585de7b7b6c") {
    label,
    expires,
    hash,
    customer_email,
    customer_id,
    profile_id,
    payment_id,
    method,
    active,
    created_at,
    updated_at,
    last_use,
    address {
      region {
        region_code,
        region,
        region_id
      },
      region_id,
      country_id,
      street,
      company,
      telephone,
      postcode,
      city,
      firstname,
      lastname
    },
    additional {
      cc_type,
      cc_bin,
      cc_last4,
      cc_exp_year,
      cc_exp_month
    }
  }
}
```

Example response:

```
{
  "data": {
    "tokenBaseCards": [
      {
        "label": "Visa XXXX-0027",
        "expires": "2022-12-31 23:59:59",
        "hash": "ec431a3e1f9904a35dc083a257cf2585de7b7b6c",
        "customer_email": "roni_cost@example.com",
        "customer_id": 1,
        "profile_id": null,
        "payment_id": "9F6B429B2C2A64D5E05341588E0A3F70",
        "method": "paradoxlabs_cybersource",
        "active": true,
        "created_at": "2020-02-25 18:05:12",
        "updated_at": "2020-02-25 18:05:12",
        "last_use": null,
        "address": {
          "region": {
            "region_code": "PA",
            "region": "Pennsylvania",
            "region_id": 51
          },
          "region_id": 51,
          "country_id": "US",
          "street": [
            "123 Test Lane",
            ""
          ],
          "company": "",
          "telephone": "1111111111",
          "postcode": "17603",
          "city": "Test City",
          "firstname": "John",
          "lastname": "Doe"
        },
        "additional": {
          "cc_type": "VI",
          "cc_bin": "400700",
          "cc_last4": "0027",
          "cc_exp_year": "2022",
          "cc_exp_month": "12"
        }
      }
    ]
  }
}
```


Fetch checkout config

Example request:

```
{
  tokenBaseCheckoutConfig(method:"paradoxlabs_cybersource") {
    method
,
    useVault,
    canSaveCard,
    forceSaveCard,
    defaultSaveCard,
    logoImage,
    requireCcv,
    canStoreBin,
    availableTypes {key, value},
    months {key, value},
    years {key, value},
    hasVerification,
    cvvImageUrl,
    cardinalAuthUrl,
    cardinalJWT,
    cardinalScript,
    fingerprintUrl,
    paramUrl
  }
}
```

Example response:

```
{
  "data": {
    "tokenBaseCheckoutConfig": {
      "method": "paradoxlabs_cybersource",
      "useVault": true,
      "canSaveCard": true,
      "forceSaveCard": false,
      "defaultSaveCard": true,
      "logoImage": "false",
      "requireCcv": false,
      "canStoreBin": null,
      "availableTypes": [
        {
          "key": "AE",
          "value": "American Express"
        },
        ...
      ],
      "months": [
        {
          "key": "1",
          "value": "01 - January"
        },
        ...
      ],
      "years": [
        {
          "key": "2020",
          "value": "2020"
        },
        ...
      ],
      "hasVerification": true,
      "cvvImageUrl":
"https://example.com/pub/static/version1608830907/graphql/_view/en_US/Magento_Checkout/cvv.png",
      "cardinalAuthUrl": "https://example.com/pdl_cybs/cardinalCruise/getAuthPayload/",
      "cardinalJWT": "eyJhbGciOiJIUzI1NiIsInR5...y2q8",
      "cardinalScript": "https://songbirdstag.cardinalcommerce.com/edge/v1/songbird.js",
      "fingerprintUrl": "https://h.online-metrix.net/fp/tags.js?...",
      "paramUrl": "https://example.com/pdl_cybs/secureAccept/getParams/"
    }
  }
}
```


Create card

Example request:

```
mutation {
  createTokenBaseCard(
    input: {
      expires: "2022-12-31 23:59:59",
      customer_ip: "127.0.0.1",
      customer_email: "roni_cost@example.com",
      payment_id: "9F6B429B2C2A64D5E05341588E0A3F70",
      method: "paradoxlabs_cybersource",
      active: true,
      address: {
        region: {
          region_code: "PA",
          region: "Pennsylvania",
          region_id: 51
        },
        country_id: US,
        street: [
          "123 Test St.",
          "Apt 9"
        ],
        company: "",
        telephone: "111-111-1111",
        postcode: "12345",
        city: "Testcity",
        firstname: "John",
        lastname: "Doe"
      },
      additional: {
        cc_type: "VI",
        cc_last4: "0027",
        cc_exp_year: "2022",
```

```
        cc_exp_month: "12",
        cc_bin: "400700"
      }
    }
  ) {
    label,
    expires,
    hash,
    customer_email,
    customer_id,
    customer_ip,
    payment_id,
    method,
    active,
    created_at,
    updated_at,
    last_use,
    address {
      region {
        region_code,
        region,
        region_id
      },
      region_id,
      country_id,
      street,
      company,
      telephone,
      postcode,
      city,
      firstname,
      lastname
    },
    additional {
      cc_type,
      cc_bin,
      cc_last4,
      cc_exp_year,
      cc_exp_month
    }
  }
}
```

Example response:

```
{
  "data": {
    "createTokenBaseCard": {
      "label": "Visa XXXX-0027",
      "expires": "2022-12-31 23:59:59",
      "hash": "a5412c321a5de0c1f3964492e0bfba2b48e5984b",
      "customer_email": "roni_cost@example.com",
      "customer_id": 1,
      "customer_ip": "127.0.0.1",
      "payment_id": "qqqqqq",
      "method": "paradoxlabs_cybersource",
      "active": true,
      "created_at": null,
      "updated_at": null,
      "last_use": null,
      "address": {
        "region": {
          "region_code": "PA",
          "region": "Pennsylvania",
          "region_id": 51
        },
        "region_id": 51,
        "country_id": "US",
        "street": [
          "123 Test St.",
          "Apt 9"
        ],
        "company": "",
        "telephone": "111-111-1111",
        "postcode": "12345",
        "city": "Testcity",
        "firstname": "John",
        "lastname": "Doe"
```

```
      },
      "additional": {
        "cc_type": "VI",
        "cc_bin": "400700",
        "cc_last4": "0027",
        "cc_exp_year": "2022",
        "cc_exp_month": "12"
      }
    }
  }
}
```

Delete card

Example request:

```
mutation {
  deleteTokenBaseCard(hash:"88bb7dc06faad55c77177446ed83047811234008")
}
```

Example response:

```
{
  "data": {
    "deleteTokenBaseCard": true
  }
}
```

Get Secure Acceptance form

Example request:

```
query {
  cyberSourceSecureAcceptParams(input: {
    cartId: "kDyOEKkJmIOxa6H3ceus6MjMaSF9lao1"
    source: checkout
  }) {
    iframeAction
    iframeParams
  }
}
```

Example response:

```
{
  "data": {
    "cyberSourceSecureAcceptParams": {
      "iframeAction": "https://testsecureacceptance.cybersource.com/embedded/token/create",
      "iframeParams": "{\"access_key\":\"c0083be4a1d631699612345285072a96\",\"locale\":\"en-
us\",\"payment_method\":\"card\",\"profile_id\":\"8F003709-7AA5-4D64-8AFC-
B1F8374B18B2\",\"reference_number\":\"620d02077398c0.58624424\",\"signed_date_time\":\"2022-02-
16T13:54:15Z\",\"skip_decision_manager\":\"false\",\"transaction_uuid\":\"620d02077398c0.58624424\",\"consumer
_id\":\"2\",\"merchant_defined_data99\":\"VaaFv9jm0OERMpQTCHJSRA5QDOoeJjwb\",\"merchant_defined_data100\":null
,\"override_custom_receipt_page\":\"https:\\/\\/example.com\\/pdl_cybs\\/secureAccept\\/complete\\/\",\"partne
r_solution_id\":\"DEQXVEEG\",\"signed_field_names\":\"access_key,locale,payment_method,profile_id,reference_nu
mber,signed_date_time,skip_decision_manager,transaction_uuid,consumer_id,merchant_defined_data99,merchant_defi
ned_data100,override_custom_receipt_page,partner_solution_id,signed_field_names,customer_ip_address,transactio
n_type,amount,currency,bill_to_forename,bill_to_surname,bill_to_email,bill_to_company_name,bill_to_address_cou
ntry,bill_to_address_city,bill_to_address_state,bill_to_address_line1,bill_to_address_line2,bill_to_address_po
stal_code,bill_to_phone\",\"customer_ip_address\":\"127.0.0.1\",\"transaction_type\":\"create_payment_token\",
\"amount\":0,\"currency\":\"USD\",\"bill_to_forename\":\"John\",\"bill_to_surname\":\"Doe\",\"bill_to_email\":
\"email@example.com\",\"bill_to_company_name\":\"company\",\"bill_to_address_country\":\"US\",\"bill_to_addres
s_city\":\"Lancaster\",\"bill_to_address_state\":\"PA\",\"bill_to_address_line1\":\"8 N Queen
St\",\"bill_to_address_line2\":\"\",\"bill_to_address_postal_code\":\"17603\",\"bill_to_phone\":\"123-456-
0000\",\"signature\":\"sMBz2FVY7uD7c9tNcZblD92X\\/zXUGaROKlzTpgvzkwE=\"}"
    }
  }
}
```

Get Cardinal Cruise/Payer Authentication params

Example request:

```
query {
  cyberSourceCardinalCruiseAuthPayload(input: {
    cartId: "kDyOEKkJmIOxa6H3ceus6MjMaSF9lao1"
  }) {
    JWT
    authPayload
    orderPayload
  }
}
```

Example response:

```
{
  "data": {
    "cyberSourceCardinalCruiseAuthPayload": {
      "JWT":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJqdGkiOiI2MWViMGJhNTA1Zjh…6HPbmHsuVnHerjKVcjhhV7gxSiHRxkHslo",
      "authPayload":
"{\"AcsUrl\":\"https:\\/\\/merchantacsstag.cardinalcommerce.com\\/MerchantACSWeb\\/pareq.jsp?vaa=b&gold=AAA…AA
AAAAAAAAAAAAAAAAAAAAAAAA\",\"Payload\":\"eNpVUstuwjAQvPsrEOo5jp2ACFos0VIJUItoC6Ti5jpWE0oe2AmPfn3thJTWp531jndn
1rCKlZSTNykqJRk8…D1CqwH\"}",
      "orderPayload":
"{\"OrderDetails\":{\"TransactionId\":\"ruWZHDW2aq2hhscQIoZ0\",\"OrderNumber\":\"000000837\"}}"
    }
  }
}
```

Place an order

Example request:

```
mutation {
  setPaymentMethodOnCart(
    input: {
      cart_id: "kDyOEKkJmIOxa6H3ceus6MjMaSF9lao1"
      payment_method: {
        code: "paradoxlabs_cybersource",
        tokenbase_data: {
          card_id: "ec431a3e1f9904a35dc083a257cf2585de7b7b6c",
          cc_cid: "123",
          save: true,
          response_jwt: ""
        }
      }
    }
  ) {
    cart {
      selected_payment_method {
        code,
        tokenbase_card_id,
        tokenbase_data {
          cc_type,
          cc_last4,
          cc_exp_year,
          cc_exp_month
        }
      }
    }
  }
  placeOrder(
    input: {
      cart_id: "kDyOEKkJmIOxa6H3ceus6MjMaSF9lao1"
    }
  ) {
    order {
      order_number
    }
  }
}
```

Example response:

```
{
  "data": {
    "setPaymentMethodOnCart": {
```

```
      "cart": {
        "selected_payment_method": {
          "code": "paradoxlabs_cybersource",
          "tokenbase_card_id": "ec431a3e1f9904a35dc083a257cf2585de7b7b6c",
          "tokenbase_data": {
            "cc_type": "VI",
            "cc_last4": "0027",
            "cc_exp_year": "2022",
            "cc_exp_month": "12"
          }
        }
      }
    },
    "placeOrder": {
      "order": {
        "order_number": "000000676"
      }
    }
  }
}
```

How-To: API Checkout Flow

The checkout flow for this extension is a little complex, due to the security mechanisms involved. There are essentially two isolated pieces: Initializing and using Secure Acceptance to display the credit card form and store a new credit card; then loading stored credit cards, selecting one, and submitting it with the order. If 3D Secure support is required, card authentication will also have to be implemented around the order submission, by means of the Cardinal Commerce Songbird library and API.

Step 1: Fetch checkout payment parameters

For GraphQL, see the '*Fetch checkout config*' example. For standard checkout's REST requests, these parameters are provided to JS by Magento's frontend layout system.

There are several parameters that are critical to the CyberSource checkout flow. Most important is the Secure Acceptance params URL:

```
      "paramUrl": "https://example.com/pdl_cybs/secureAccept/getParams/"
```

If you intend to implement 3D Secure, you will also require the fingerprint script URL and cardinal parameters:

```
      "cardinalAuthUrl": "https://example.com/pdl_cybs/cardinalCruise/getAuthPayload/",
      "cardinalJWT": "eyJhbGciOiJIUzI1NiIsInR5...y2q8",
      "cardinalScript": "https://songbirdstag.cardinalcommerce.com/edge/v1/songbird.js",
      "fingerprintUrl": "https://h.online-metrix.net/fp/tags.js?...",
```

The other parameters communicate Magento payment settings and state, and can be used or disregarded as you choose.

Step 2: Fetch Secure Acceptance Payload

When the time comes to initiate the credit card form, you will need to query `cyberSourceSecureAcceptParams`. We do this any time the 'new card' option is selected on our payment method, or payment method visibility changes, or the Secure Acceptance form is completed or reset.

For GraphQL, see the *'Get Secure Acceptance form'* example. The query takes many of the same inputs, but the keys differ. The response has iframeParams JSON-encoded. See the example and GraphQL schema for specifics.

For REST, your POST request must include the following request parameters as payload:

```
source=checkout
guest_email=
billing[firstname]=
billing[lastname]=
billing[company]=
billing[street][]=
billing[city]=
billing[regionCode]=
billing[region]=
billing[postcode]=
billing[countryId]=
billing[telephone]=
```

If there is not an active logged-in session with Magento, you should include guest_email with the user's email address (otherwise, email will be pulled from the logged-in customer). You may also include card_id (for updating an existing stored card) and billing (for providing an address separate from the quote).

Billing address must not be modified after initializing the Secure Acceptance form.

If successful, this request will return a JSON payload like:


```
{
    "iframeAction": "https:\/\/testsecureacceptance.cybersource.com\/embedded\/token\/create",
    "iframeParams": {
        "access_key": "a89abf0...9150b",
        "locale": "en-us",
        "payment_method": "card",
        "profile_id": "8F003709-1111-1111-1111-B1F81111E8B2",
        "reference_number": "5fed057c11b2f8.98581327",
        "signed_date_time": "2020-12-30T22:55:56Z",
        "skip_decision_manager": "true",
        "transaction_uuid": "5fed057c11b2f8.98581327",
        "consumer_id": "1",
        "merchant_defined_data99": "j5nfpi3f8aiema9od709qsnsdo",
        "merchant_defined_data100": "checkout",
        "override_custom_receipt_page": "https:\/\/example.com\/pdl_cybs\/secureAccept\/complete\/",
        "partner_solution_id": "DEQXVEEG",
        "signed_field_names": "access_key,...,bill_to_phone",
        "customer_ip_address": "127.0.0.1",
        "transaction_type": "create_payment_token",
        "amount": 0,
        "currency": "USD",
        "bill_to_forename": "Veronica",
        "bill_to_surname": "Costello",
        "bill_to_email": "roni_cost@example.com",
        "bill_to_company_name": "",
        "bill_to_address_country": "US",
        "bill_to_address_city": "Calder",
        "bill_to_address_state": "MI",
        "bill_to_address_line1": "6146 Honey Bluff Parkway",
        "bill_to_address_line2": "",
        "bill_to_address_postal_code": "49628-7978",
        "bill_to_phone": "(555) 229-3326",
        "signature": "LPsi8yHK+uPOSkRAOXu\/IrnuOTlmHbAWwj7mw9xKr48="
    }
}
```


If an error occurs, it will return a 400 response code, and may provide a JSON response including error message such as:

```
{
    "message": "Invalid email address."
}
```

Step 3: Initialize Secure Acceptance

Once you have the Secure Acceptance payload (`iframeAction` and `iframeParams`), you need to initialize Secure Acceptance itself. This requires creating an iframe on the page to contain the Secure Acceptance form; creating a

form that targets that iframe; setting the form action to `iframeAction`; filling that form with the `iframeParams`; then submitting it, to initiate a browser POST to Secure Acceptance.

Our checkout process hardcodes the iframe, but generates the form on the fly. It does this as such:

```
var form = document.createElement('form');
form.target = 'paradoxlabs_cybersource_iframe';
form.method = 'post';
form.action = data.iframeAction;

for (var key in data.iframeParams) {
    var input = document.createElement('input');
    input.type = 'hidden';
    input.name = key;
    input.value = data.iframeParams[key];
    form.appendChild(input);
}

document.body.appendChild(form);
form.submit();
```

You will also need the page to contain an `input#paradoxlabs_cybersource-communicator` element, to receive updates on the Secure Acceptance process. Secure Acceptance will redirect to a page on the Magento installation (`override_custom_receipt_page`: https://example.com/pdl_cybs/secureAccept/complete/); that page will store the card, and output JavaScript which writes the card data to `input#paradoxlabs_cybersource-communicator`. This allows the checkout JS to observe that to discover (1) when the user has completed the Secure Acceptance process; and (2) what the new stored card data is (including its card ID, type, last4, etc.).

Your page must be manipulable by JS executed on that target page, or you will not be able to receive event updates. If the communicator page and the parent frame have different domains or subdomains, this will require setting appropriate CORS (cross-origin resource sharing) headers on responses from your Magento installation.

Step 4: Receive Secure Acceptance success

If successful, your `input#paradoxlabs_cybersource-communicator` element will be updated with a JSON payload value:

```
{
    “success”: true,
    “canceled”: false,
    “error”: “”,
    “card”: {
        “id”: “ec431a3e1f9904a35dc083a257cf2585de7b7b6c”,
        “label”: “VI XXXX-0027”,
        “selected”: false,
        “new”: true,
        “type”: “VI”,
        “cc_bin”: “400000”,
        “cc_last_4”: “0027”
    }
}
```

If unsuccessful, you’ll receive a communication payload like:

```
{
    “success”: false,
    “canceled”: false,
    “error”: “An error occurred.”,
    “card”: []
}
```

You should take this new-card info and add it to whatever interface you have for presenting existing stored cards, with the newly stored card selected, and hide all Secure Acceptance-related interfaces. In our interfaces, we use a dropdown of stored cards (plus a 'Add New Card' option that triggers Secure Acceptance step 1 above), and inject the newly stored card at the bottom.

At this time the user's credit card is stored and usable for placing an order, but will not be visible on their account unless submitted through checkout (with save=1) or otherwise marked active by API.

At this time, the customer should confirm their payment details (select a credit card to pay by), then hit 'Place Order'.

Step 5: Submit checkout

When the customer places their order, you must transmit their selected stored card hash `card_id` to Magento, plus the CVV if necessary. If CVV is enabled, the card ID and CVV **must** be sent to Magento in exactly the same request as the final place-order command. CVV values are not stored under any circumstances—sending it on a previous request would be equivalent to not sending the CVV at all.

For **GraphQL**, see the *'Get Secure Acceptance* form

Example request:

```
query {
  cyberSourceSecureAcceptParams(input: {
    cartId: "kDyOEKkJmIOxa6H3ceus6MjMaSF9lao1"
    source: checkout
  }) {
    iframeAction
    iframeParams
  }
}
```

Example response:


```
{
  "data": {
    "cyberSourceSecureAcceptParams": {
      "iframeAction": "https://testsecureacceptance.cybersource.com/embedded/token/create",
      "iframeParams": "{\"access_key\":\"c0083be4a1d631699612345285072a96\",\"locale\":\"en-
us\",\"payment_method\":\"card\",\"profile_id\":\"8F003709-7AA5-4D64-8AFC-
B1F8374B18B2\",\"reference_number\":\"620d02077398c0.58624424\",\"signed_date_time\":\"2022-02-
16T13:54:15Z\",\"skip_decision_manager\":\"false\",\"transaction_uuid\":\"620d02077398c0.58624424\",\"consumer
_id\":\"2\",\"merchant_defined_data99\":\"VaaFv9jm0OERMpQTCHJSRA5QDOoeJjwb\",\"merchant_defined_data100\":null
,\"override_custom_receipt_page\":\"https:\\/\\\/example.com\\/pdl_cybs\\/secureAccept\\/complete\\/\",\"partne
r_solution_id\":\"DEQXVEEG\",\"signed_field_names\":\"access_key,locale,payment_method,profile_id,reference_nu
mber,signed_date_time,skip_decision_manager,transaction_uuid,consumer_id,merchant_defined_data99,merchant_defi
ned_data100,override_custom_receipt_page,partner_solution_id,signed_field_names,customer_ip_address,transactio
n_type,amount,currency,bill_to_forename,bill_to_surname,bill_to_email,bill_to_company_name,bill_to_address_cou
ntry,bill_to_address_city,bill_to_address_state,bill_to_address_line1,bill_to_address_line2,bill_to_address_po
stal_code,bill_to_phone\",\"customer_ip_address\":\"127.0.0.1\",\"transaction_type\":\"create_payment_token\",
\"amount\":0,\"currency\":\"USD\",\"bill_to_forename\":\"John\",\"bill_to_surname\":\"Doe\",\"bill_to_email\":
\"email@example.com\",\"bill_to_company_name\":\"company\",\"bill_to_address_country\":\"US\",\"bill_to_addres
s_city\":\"Lancaster\",\"bill_to_address_state\":\"PA\",\"bill_to_address_line1\":\"8 N Queen
St\",\"bill_to_address_line2\":\"\",\"bill_to_address_postal_code\":\"17603\",\"bill_to_phone\":\"123-456-
0000\",\"signature\":\"sMBz2FVY7uD7c9tNcZblD92X\\/zXUGaROKlzTpgvzkwE=\"}"
    }
  }
}
```


Get Cardinal Cruise/Payer Authentication params

Example request:

```
query {
  cyberSourceCardinalCruiseAuthPayload(input: {
```

```
    cartId: "kDyOEKkJmIOxa6H3ceus6MjMaSF9lao1"
  }) {
    JWT
    authPayload
    orderPayload
  }
}
```

Example response:

```
{
  "data": {
    "cyberSourceCardinalCruiseAuthPayload": {
      "JWT":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJqdGkiOiI2MWViMGJhNTA1Zjh…6HPbmHsuVnHerjKVcjhhV7gxSiHRxkHslo",
      "authPayload":
"{\"AcsUrl\":\"https:\\/\\/merchantacsstag.cardinalcommerce.com\\/MerchantACSWeb\\/pareq.jsp?vaa=b&gold=AAA…AA
AAAAAAAAAAAAAAAAAAAAAAAAA\",\"Payload\":\"eNpVUstuwjAQvPsrEOo5jp2ACFos0VIJUItoC6Ti5jpWEOoe2AmPfn3thJTWp531jndn
1rCKlZSTNykqJRk8…D1CqwH\"}",
      "orderPayload":
"{\"OrderDetails\":{\"TransactionId\":\"ruWZHDW2aq2hhscQIoZ0\",\"OrderNumber\":\"000000837\"}}"
    }
  }
}
```

Place an order' example request and response. Note that the selected card hash is sent as `payment_method.tokenbase_data.card_id`, and the CVV is sent as `payment_method.tokenbase_data.cc_cid`. If the card should be saved and visible for reuse, also send `payment_method.tokenbase_data.save: true`. All other information is already on file with the stored card, thanks to the Secure Acceptance flow completed in steps 1-4.

For **REST**, see Magento documentation for the /carts/mine/payment-information endpoint, or view the API request sent for the same endpoint when going through actual Magento checkout (which uses the REST API). Note that `paymentMethod.additional_data` should contain the same payload data described above:

```
{
    "paymentMethod": {
        "method": "paradoxlabs_cybersource",
        "additional_data": {
            "card_id": "ec431a3e1f9904a35dc083a257cf2585de7b7b6c",
            "cc_cid": "123",
            "save": true
        }
    }
}
```

Also note, if you intend to support 3D Secure, that requires additional work as detailed below.

How-To: API 3D Secure Flow

Implementing 3D Secure follows exactly the same flow as noted above for standard Secure Acceptance checkout, plus some additional work. We implement the 'Cardinal Cruise Hybrid' flow, integrating Cardinal Commerce's Songbird.js validation library with CyberSource's built-in backend processing of the results. See: https://cardinaldocs.atlassian.net/wiki/spaces/CC/pages/360668/Cardinal+Cruise+Hybrid

Initialize Cardinal Cruise JS

After Step 1 of the standard flow, you should have checkout config parameters including the following four:

```
      "cardinalAuthUrl": "https://example.com/pdl_cybs/cardinalCruise/getAuthPayload/",
      "cardinalJWT": "eyJhbGciOiJIUzI1NiIsInR5...y2q8",
      "cardinalScript": "https://songbirdstag.cardinalcommerce.com/edge/v1/songbird.js",
      "fingerprintUrl": "https://h.online-metrix.net/fp/tags.js?...",
```

The `fingerprintUrl` should be loaded on the page as a JS script. No further attention to it is needed, provided the script loads and executes successfully.

The `cardinalScript` should be loaded on the page, with a callback to initialize the Cardinal JS once it has loaded. The callback should register a validation callback, then initialize setup, passing along the `cardinalJWT` payload from config. We do this all as such on checkout:

```
loadPayerAuth: function() {
    if (config.cardinalScript.length > 0 && config.cardinalJWT.length > 0) {
        var script = document.createElement('script');
        script.type = 'text/javascript';
        script.src = config.cardinalScript;
        script.addEventListener('load', this.initPayerAuth.bind(this));
        document.head.appendChild(script);
    }
},
initPayerAuth: function() {
    Cardinal.configure({
        payment: {
            displayLoading: true
        }
    });

    Cardinal.on('payments.validated', this.handlePayerAuthCompletion.bind(this));
    Cardinal.setup('init', {jwt: config.cardinalJWT});
},
```

The `handlePayerAuthCompletion` callback should check for an error and then either alert the user or proceed to submit checkout.

Register the card BIN

Songbird.js requires the card 'BIN' (first 6 digits) to check eligibility for 3D Secure. We do this when the customer clicks 'place order', using data for the stored card (see *Step 4: Receive Secure Acceptance success* and *Fetch card by ID*). Once you have the BIN, registering it with Songbird.js is simply:

```
Cardinal.trigger('bin.process', cc_bin);
```

Place the order

Once everything is initialized, the next step is to actually place the order. We use an iterative process, attempting the transaction and only initializing the 3D Secure interface if that attempt indicates payer authentication is required.

If this is true, your order placement (see *Step 5: Submit checkout*) will fail with an error message: *"The entered card is enrolled in Payer Authentication. Please authenticate before continuing."*

When you encounter this message, do not display it. Instead:

Request the authorization payload

For GraphQL, query cyberSourceCardinalCruiseAuthPayload to obtain the Cardinal Cruise payloads. See example *'Get Cardinal Cruise/Payer Authentication params'*. You will need to JSON decode the authPayload and orderPayload responses.

For REST, initiate a POST request to the `cardinalAuthUrl` obtained earlier from config. This should include all of the same request data as sent in *Step 2: Fetch Secure Acceptance Payload* as its payload. If successful, this will return JSON with values `authPayload`, `orderPayload`, and `JWT`.

Initiate Payer Authentication

Once you have the authorization payload, all you must do is initiate the Payer Authentication interface with those exact three values given:

```
Cardinal.continue(
    'cca',
    response.authPayload,
    response.orderPayload,
    response.JWT
);
```

This will initiate the 3D Secure verification process in a modal window, all controlled by Songbird.js. Once complete, Songbird.js will call the `handlePayerAuthCompletion` callback registered earlier during the initialization. This will either provide an error number and message, or if successful a '`responseJWT`' payload. All that is necessary is to add that to the payment data `tokenbase_data` (GraphQL) or `additional_data` (REST) as `response_jwt`, then reattempt the order submission.

```
handlePayerAuthCompletion: function(responseData, responseJWT) {
    if (responseData.ErrorNumber > 0) {
        this.responseJWT(null);

        // If Payer Auth CCA failed, throw the error message and let the user deal with it.
        alert({
            title: $.mage.__('Error'),
            content: $.mage.__(responseData.ErrorDescription)
        });
    } else {
        // If Payer Auth CCA succeeded, store the JWT and retry the order.
        this.responseJWT(responseJWT);
        this.placeOrder();
    }
},
```

Congratulations: You've placed a 3D Secure-authenticated order.

Support

If you have any questions not covered by this document, or something isn't working right, please open a ticket in our support system: support.paradoxlabs.com

Support Policy: https://store.paradoxlabs.com/support.html

License and Terms of Use: https://store.paradoxlabs.com/license.html