ParadoxLabs CyberSource Payments: User Manual

Version 4.0 – For Magento 2.4.6+ – Updated 2026-08-12

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 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 a 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.

Version 4.0 is considerably simpler to connect than earlier releases. Everything runs on the CyberSource REST API: there is one set of credentials to create, and the checkout payment form is supplied by Unified Checkout instead of a Secure Acceptance profile you have to build by hand.

If you are upgrading from 3.x, the Simple Order API certificate, the Secure Acceptance profile, and the Cardinal Cruise credentials are no longer used, and those settings have been removed from the Magento configuration. There is nothing to recreate.

The steps are the same for a sandbox account and for production, but the two are entirely separate: each has its own credentials and its own configuration. If you want to test, you must have a dedicated sandbox account. You can create one at: https://developer.cybersource.com/hello-world/sandbox.html

Also note, many CyberSource features must be enabled by CyberSource before you can use, or even see, them. If a page or an option described below is missing from your Business Center, that is a provisioning issue on their side. Contact your CyberSource representative.

What your CyberSource account needs

This integration requires the following services to be active on your account:

- **Payment processing:** card authorization, capture, and refund.
- **Token Management Service (TMS):** stores cards for reuse. Every card the extension saves is a TMS payment instrument, and this drives your ability to do partial invoicing, editing, reordering, and more.
- **Unified Checkout:** supplies the payment form used on checkout and in the customer account. Without it, the payment form cannot load.

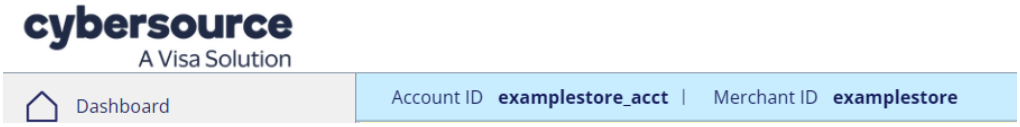
The following are optional. The extension uses them if they are enabled on your account, and skips them if they are not:

- **Payer Authentication** (3D Secure 2)
- **Decision Manager**, or Fraud Management Essentials
- **Account Updater**
- **Digital wallets:** Google Pay, Apple Pay, and Click to Pay, presented through Unified Checkout

Step 2 below shows how to confirm each of these before you spend time troubleshooting Magento.

Step 1. Find your account identifiers

Log in to the Business Center for the account you are connecting. The blue bar at the top of every page shows the two identifiers Magento needs:



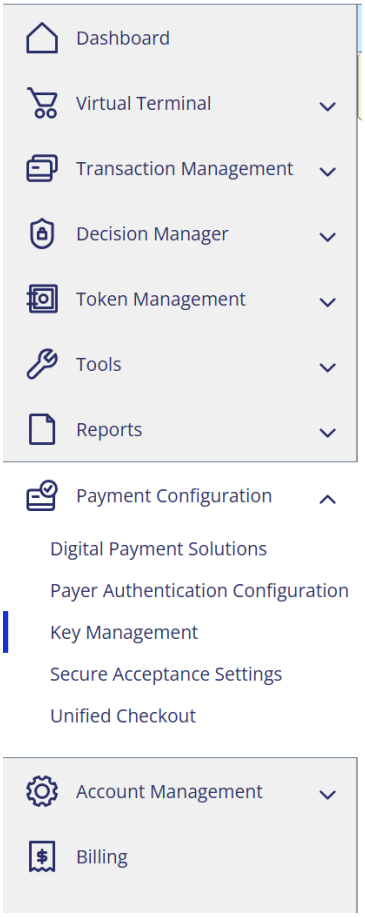
- Account ID goes into Magento as Organization ID.
- Merchant ID goes into Magento as Merchant ID.

On many accounts these two values are the same, or differ only by a suffix. Enter them exactly as shown, including case. If your login has access to more than one merchant, make sure the Merchant ID displayed is the one you intend to connect. Every page below applies to the merchant currently selected.

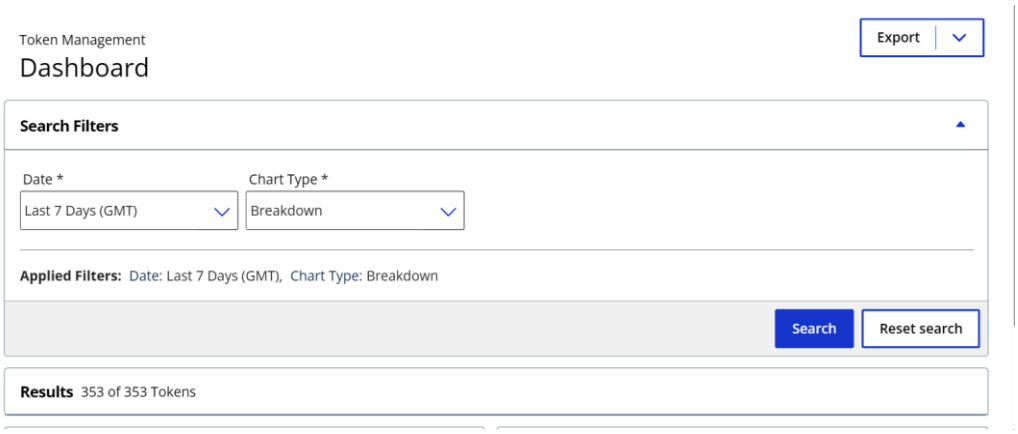
Step 2. Confirm your account is provisioned

Before entering anything into Magento, verify that the services the extension depends on are actually enabled. This is quick to check, and the most common cause of issues.

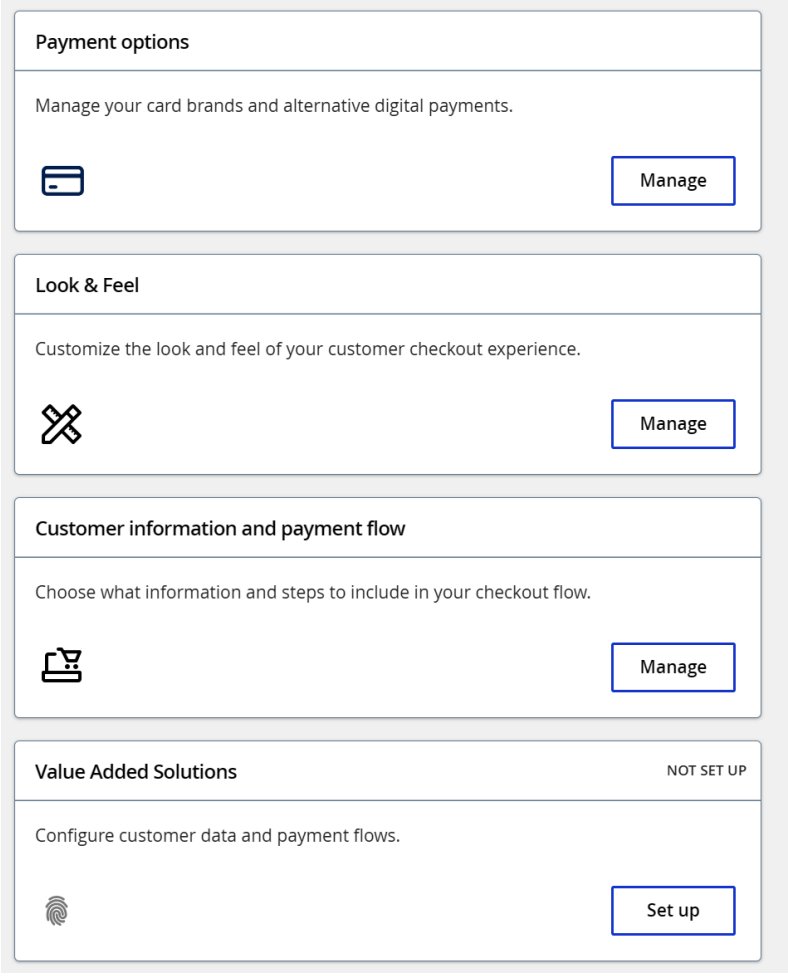
Everything to check is within the left-hand menu. If it only shows icons, expand it with the arrow at the bottom-left corner of the page.



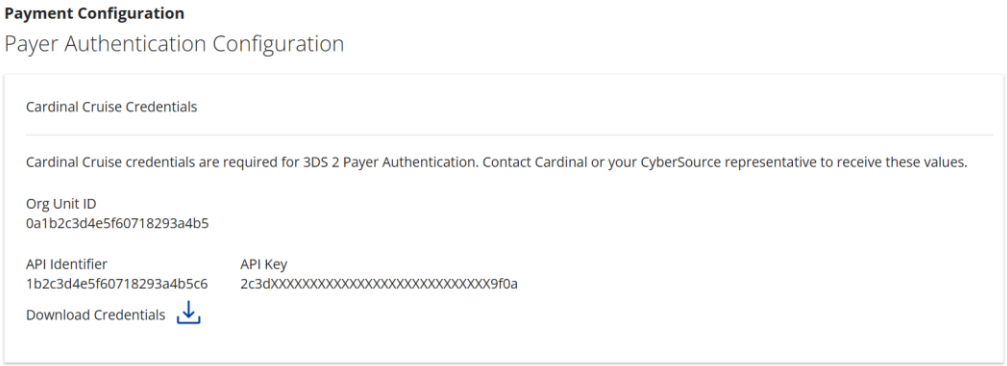
Token Management Service: Go to **Token Management > Dashboard**. If the menu entry is there and the dashboard loads, TMS is enabled on your account. A newly provisioned account will show zero tokens, which is fine — you are confirming access, not activity.



Unified Checkout: Go to **Payment Configuration > Unified Checkout**. You should get the 'My customer experience' page, with cards for Payment options, Look & Feel, and Customer information and payment flow. You will come back to this page in Step 5.

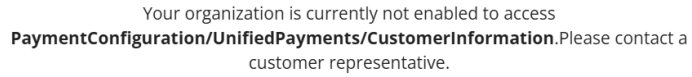


Payer Authentication (3D Secure): Go to **Payment Configuration > Payer Authentication Configuration**. If your account is set up for Payer Authentication, you should see an Org Unit ID, an API Identifier, and an API Key here.



You do not need to copy these values anywhere. Version 4.0 performs Payer Authentication over the REST API. This page proves the service is on your account. If it is empty or missing, Payer Authentication is not provisioned: either ask your CyberSource representative to enable it, or set 'Enable Payer Authentication' to 'No' in Magento.

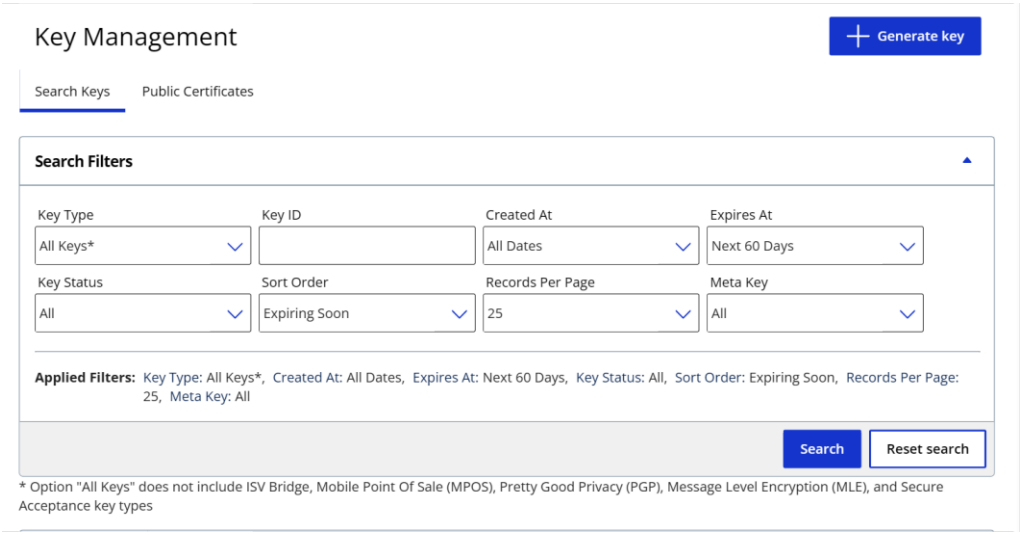
If a service is not enabled for your account, the Business Center will normally say so directly instead of showing the page:



If your account is missing Token Management Service or Unified Checkout, you must contact CyberSource Support to have it provisioned before you continue setup.

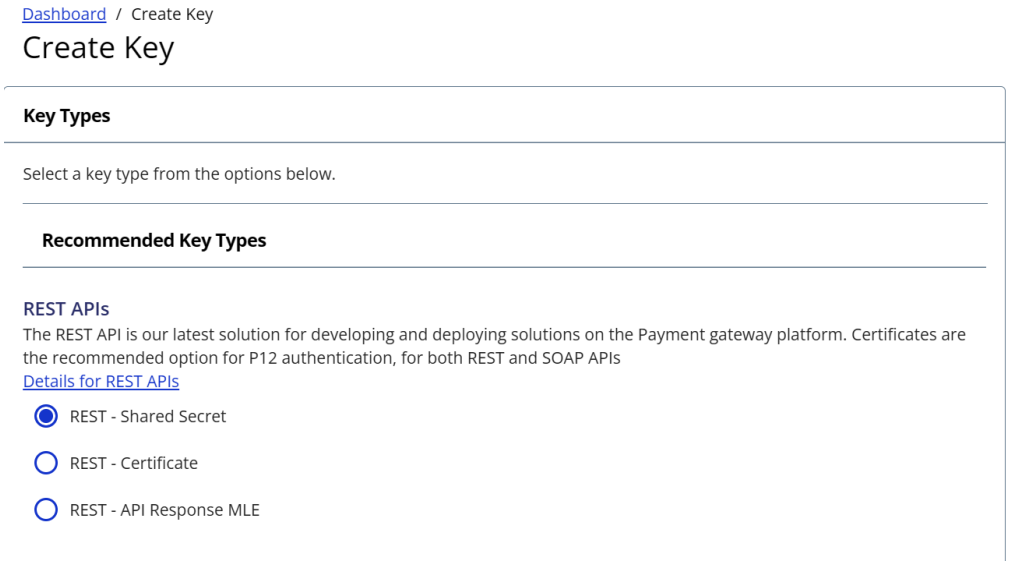
Step 3. Create your REST API key

In the Business Center, go to **Payment Configuration > Key Management**:

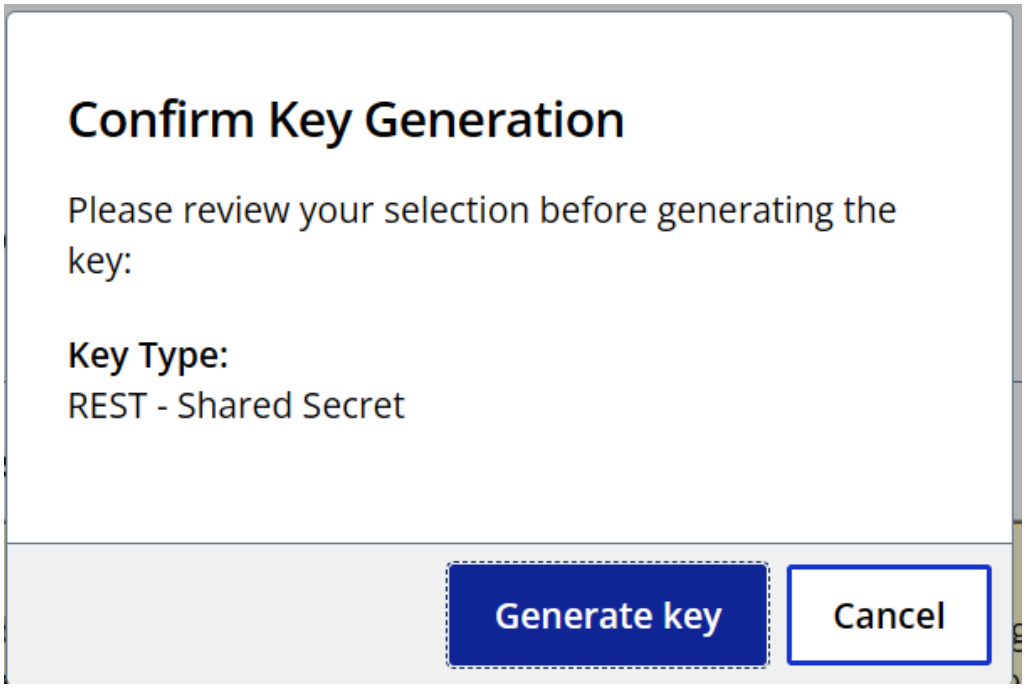


Click **Generate key** at the top right.

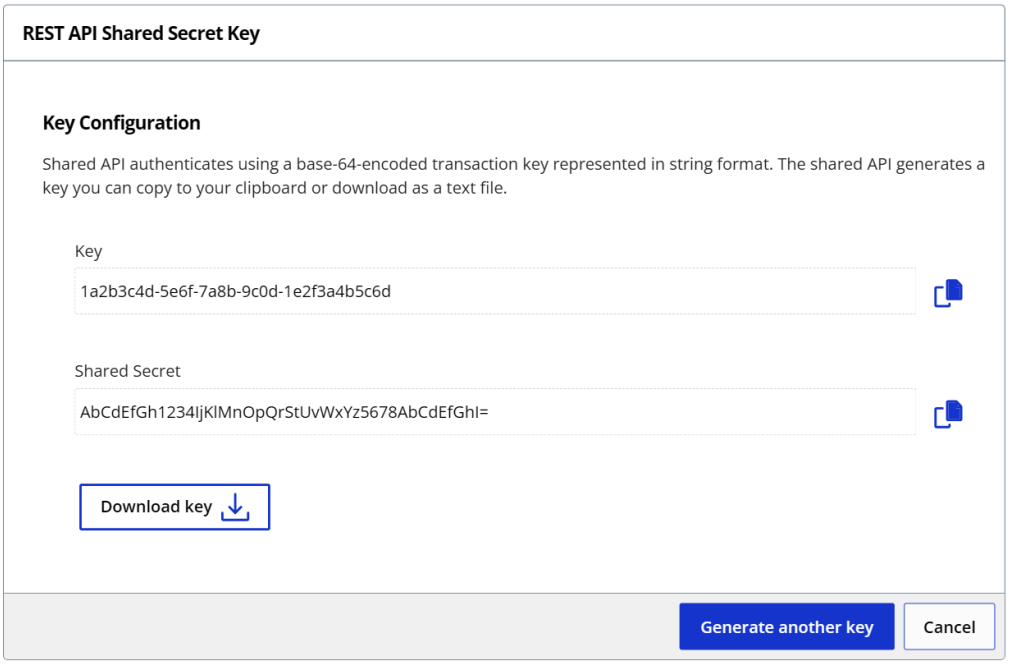
On the **Create Key** page, use the default selection: REST APIs > REST - Shared Secret. Do not select Simple Order API, Secure Acceptance, SCMP, or SOAP Toolkit; version 4.0 does not use any of them.



Scroll to the bottom of the page and click **Generate key**. A confirmation dialog appears. Check that it says REST - Shared Secret, then click Generate key again.



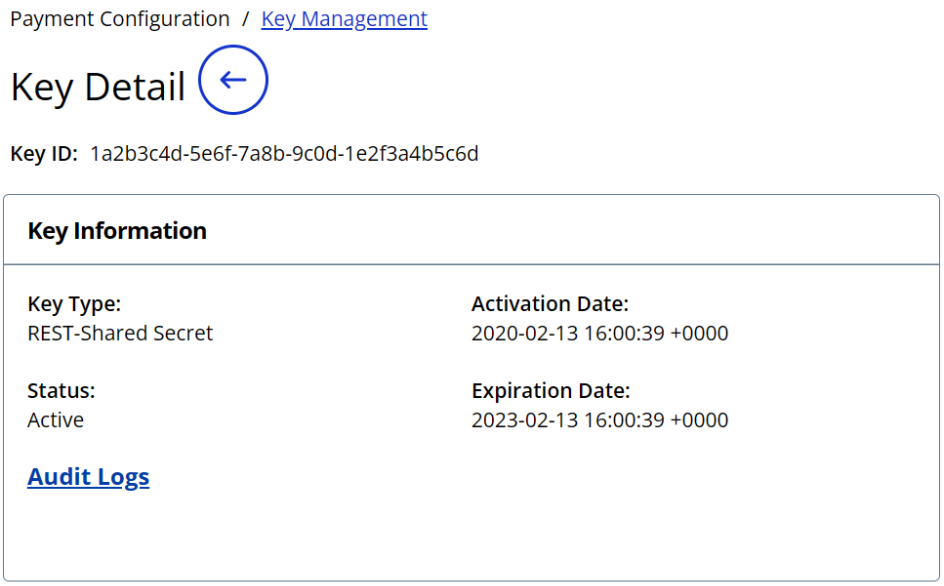
The generated key is shown once:



- **Key is your REST API Secret Key ID in Magento.** It is a UUID, in the form 1a2b3c4d-5e6f-7a8b-9c0d-1e2f3a4b5c6d.
- **Shared Secret is your REST API Secret Key in Magento.** It is a base-64 string, usually ending in '='.

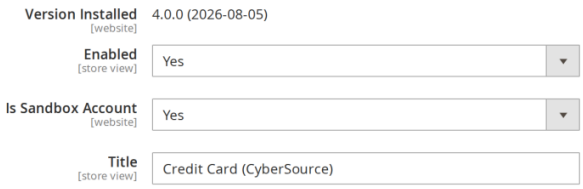
Copy both values now, or click Download key to save them to a file. The Shared Secret cannot be retrieved again once you leave this page. If you lose it, generate a new key and update Magento.

One more thing to note: REST keys expire after 3 years. Each key's detail page, reachable by clicking its Key ID in Key Management, shows an expiration date. When a key expires, every API call from Magento will start failing, so generate and install a replacement before that date. You may want to set a calendar reminder to ensure you update it before it expires.



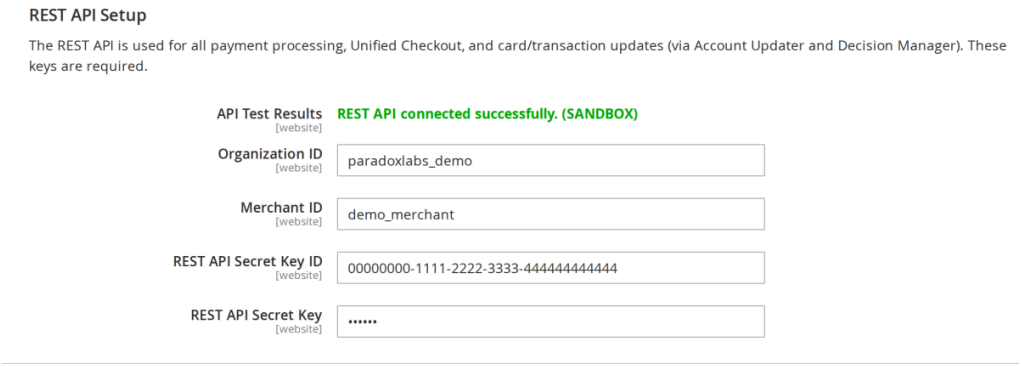
Step 4. Enter the credentials in Magento

Open your Admin Panel and go to **Stores > Settings > Configuration > Sales > Payment Methods**. Toward the bottom of the page you will find the **CyberSource** section:



- Leave 'Enabled' set to No until you have finished and tested.
- Set 'Is Sandbox Account' to Yes if these are sandbox credentials, or No for live payment processing. This must match the account your keys came from.
- Change the Title if you want to show a different label on checkout.

Then fill in the **REST API Setup** section with the four values from Steps 1 and 3, and click Save Config:

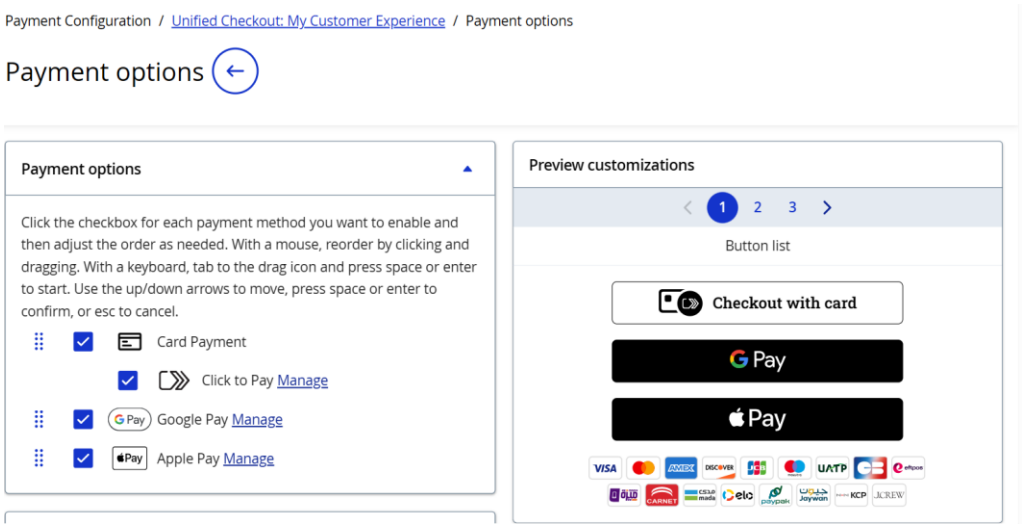


After the page reloads, API Test Results should read 'REST API connected successfully', followed by SANDBOX or PRODUCTION. If you get a red message instead, correct the problem it describes; it is almost always a mistyped or truncated key, or the wrong sandbox setting.

Step 5. Set up the Unified Checkout payment experience

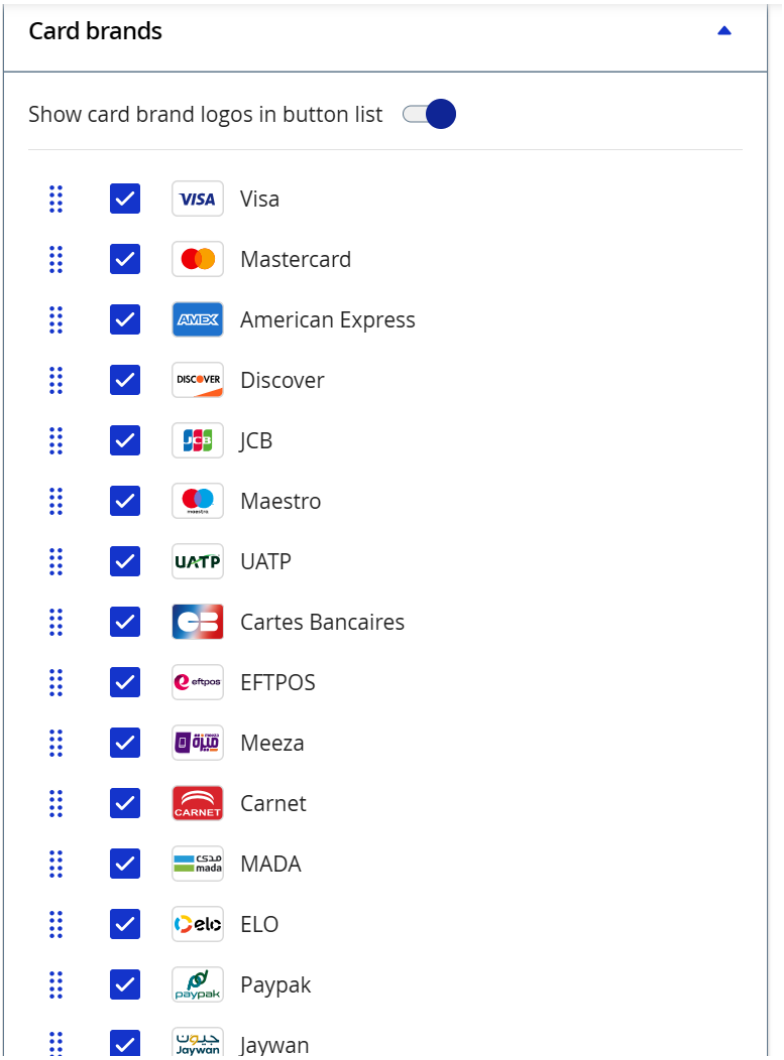
Unified Checkout replaces the Secure Acceptance profile that earlier versions required. There is no profile to create, no card types to configure, and nothing to promote. The form is configured once for your whole account. Go back to the **CyberSource EBC**, then **Payment Configuration > Unified Checkout**.

Payment options controls which payment methods and card brands your account offers:



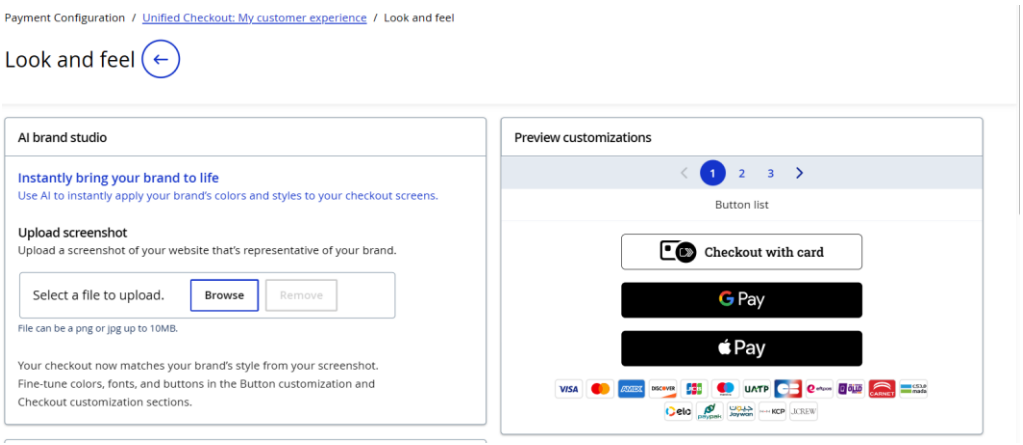
Enable Card Payment, plus any wallets you have been approved for. Google Pay, Apple Pay, and Click to Pay each have their own Manage link, with their own setup — in the case of Apple Pay, including domain verification.

Below that, Card brands controls which networks are accepted, and the order their logos appear in:



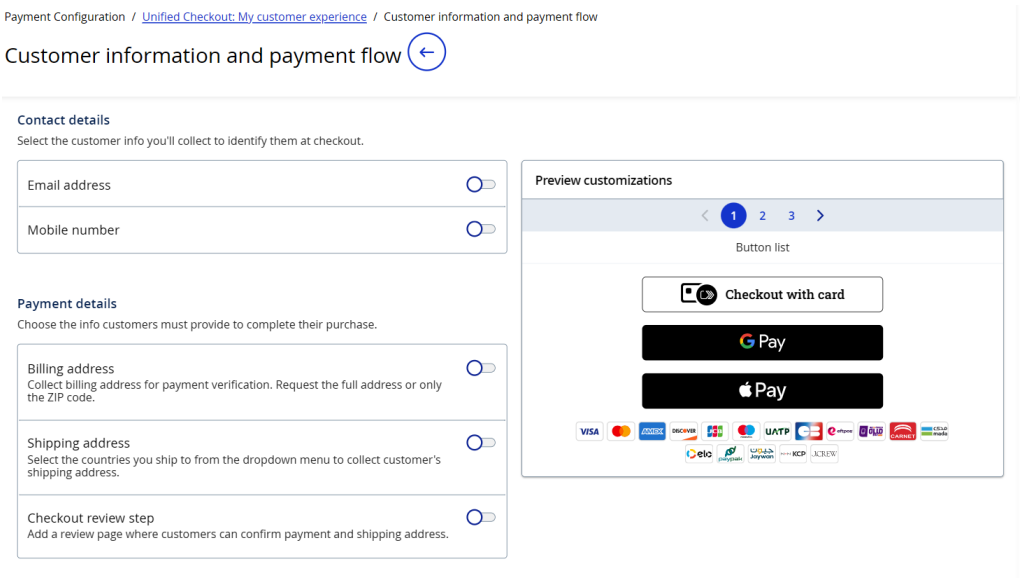
Magento has its own 'Allowed Payment Types' and 'Allowed Card Networks' settings, covered in the next section. Those narrow what this particular store requests; they cannot enable anything your CyberSource account does not already offer. Enable a brand or wallet here first, then allow it in Magento.

Look & Feel controls the appearance of the form — colors, fonts, and button styling — for every site using this account:



This is where branding work happens. Match your checkout's background and button styling so that the embedded form feels like part of it.

Customer information and payment flow controls what the form itself collects. For this integration, leave all of these switched off:

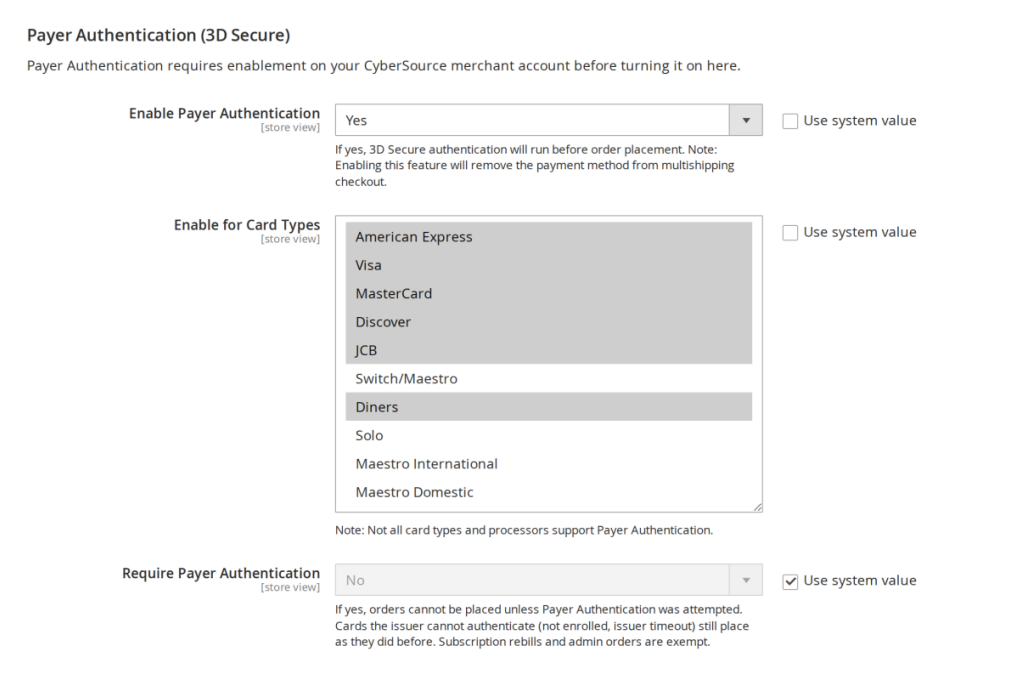


Magento already collects the email address, the billing address, and the shipping address, and provides its own order review. Turning these on asks your customers for the same information twice. If you do want a review screen inside the payment form, use Magento's 'Show Payment Review' setting instead, which is described in the next section.

Step 6. Turn on Payer Authentication (optional)

Payer Authentication is optional. If you do not want to use 3D Secure, set 'Enable Payer Authentication' to 'No' in Magento and skip this step.

Otherwise, confirm the service is on your account (done earlier in Step 2), then enable it in Magento under Payer Authentication (3D Secure):



Set 'Enable for Card Types' to the card types you want authenticated — leaving it empty disables 3D Secure for every card. 'Require Payer Authentication' controls whether an order is refused if checkout is placed without attempting authentication.

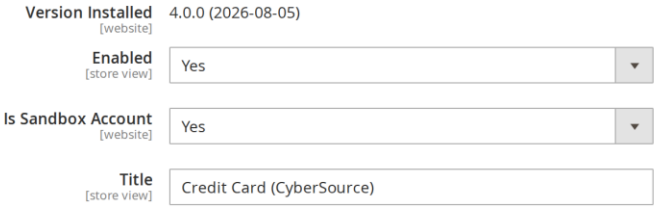
Finally, **save the settings page**.

Congratulations! Your Magento store is now connected to your CyberSource account. Before setting 'Enabled' to Yes on a live site, place a test order to confirm that the payment form loads and that a card can be charged and stored. Please see the next section for full details on the config options in Magento.

Configuration

Open your Admin Panel and go to **Stores > Settings > Configuration > Sales > Payment Methods**. Toward the bottom of the page you'll find a 'CyberSource' section. Expand it to reach all of the settings, which are grouped into the headings below.

General

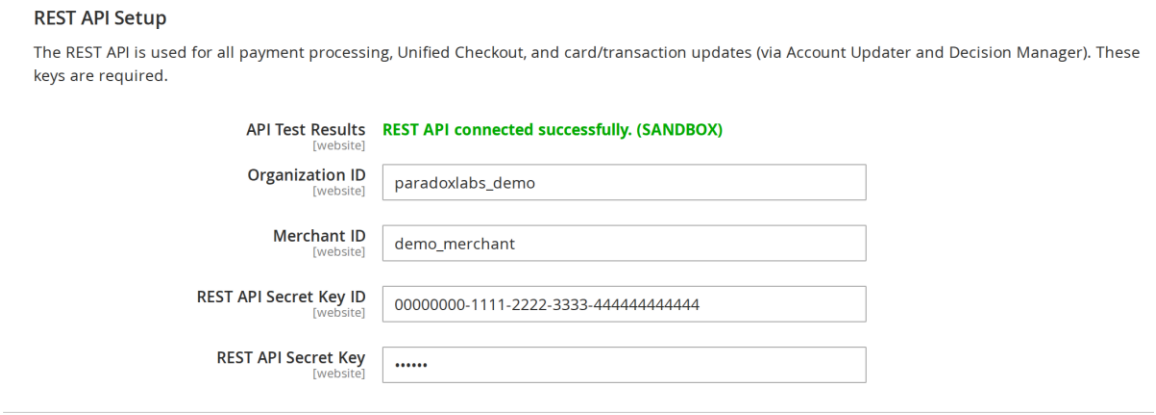


- **Version Installed**: the version of the extension currently installed. Please include this in any support requests.
- **Enabled**: Yes to offer the payment method on checkout. If disabled, you can still invoice and refund existing orders, but it will not appear as a payment option for new orders.
- **Is Sandbox Account**: if Yes, all requests are made against CyberSource's test environment and no real payments are processed. Use a CyberSource sandbox merchant account and test API keys while this is on.
- **Title**: the payment option label shown on checkout and order/status pages.

REST API Setup

The CyberSource REST API powers everything as of version 4: authorizations, captures, refunds, the Unified Checkout payment form, Payer Authentication, Account Updater, and Decision Manager. These credentials are required. (Version 4.0 removed the SOAP "Simple Order" and Secure Acceptance credentials entirely.)

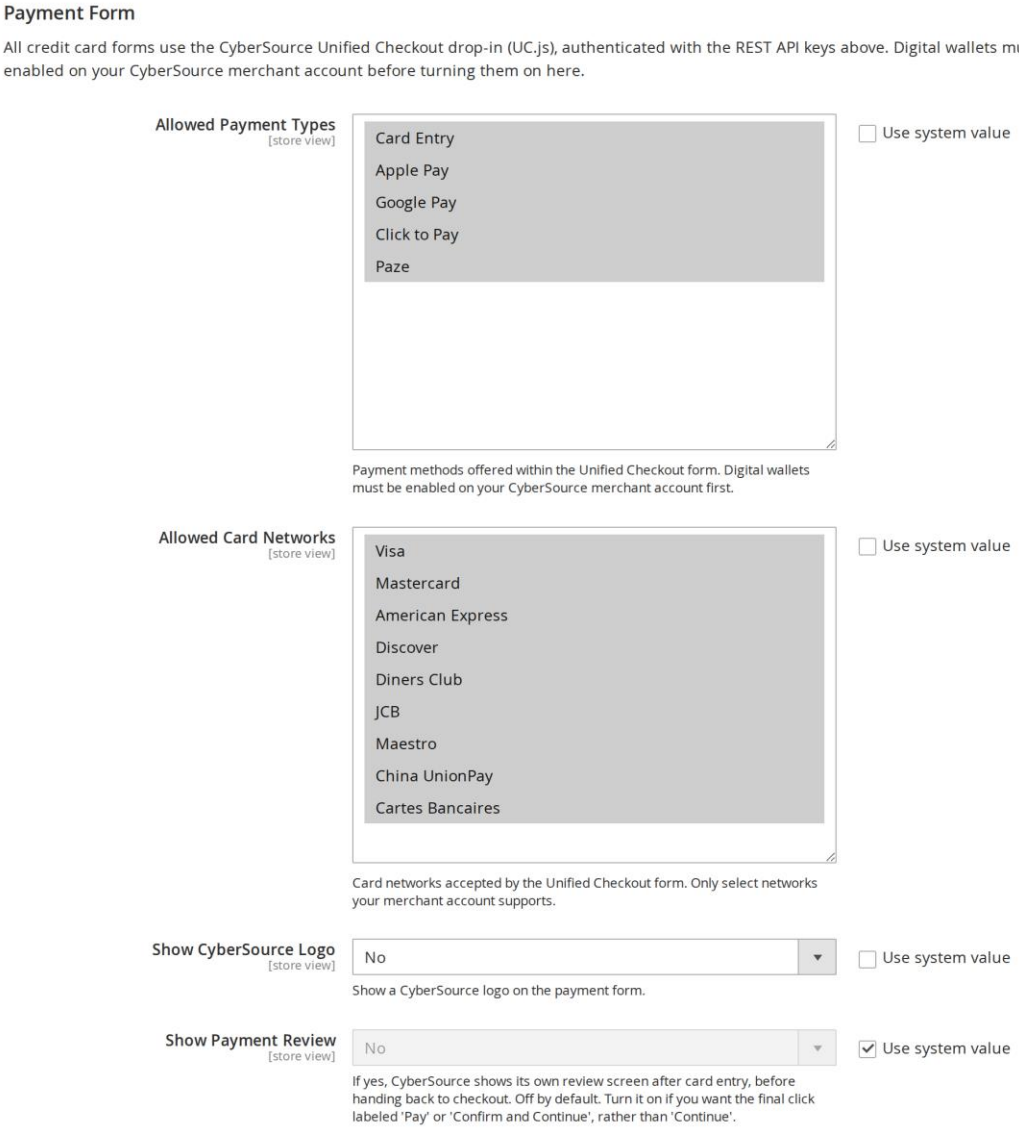
See *Connecting A New CyberSource Account: REST API Setup* for how to create and locate these keys in the CyberSource Business Center.



- **Organization ID**: your CyberSource organization (merchant) identifier.
- **Merchant ID**: the merchant ID the REST key was issued under (often the same as the Organization ID).
- **REST API Secret Key ID**: the shared-secret key identifier (a UUID) from the Business Center.
- **REST API Secret Key**: the shared-secret value paired with the key ID. Stored encrypted.
- **API Test Results**: once you save these settings, we connect to CyberSource to verify the credentials work. If the test fails, the payment method will not function until it is corrected.

Payment Form

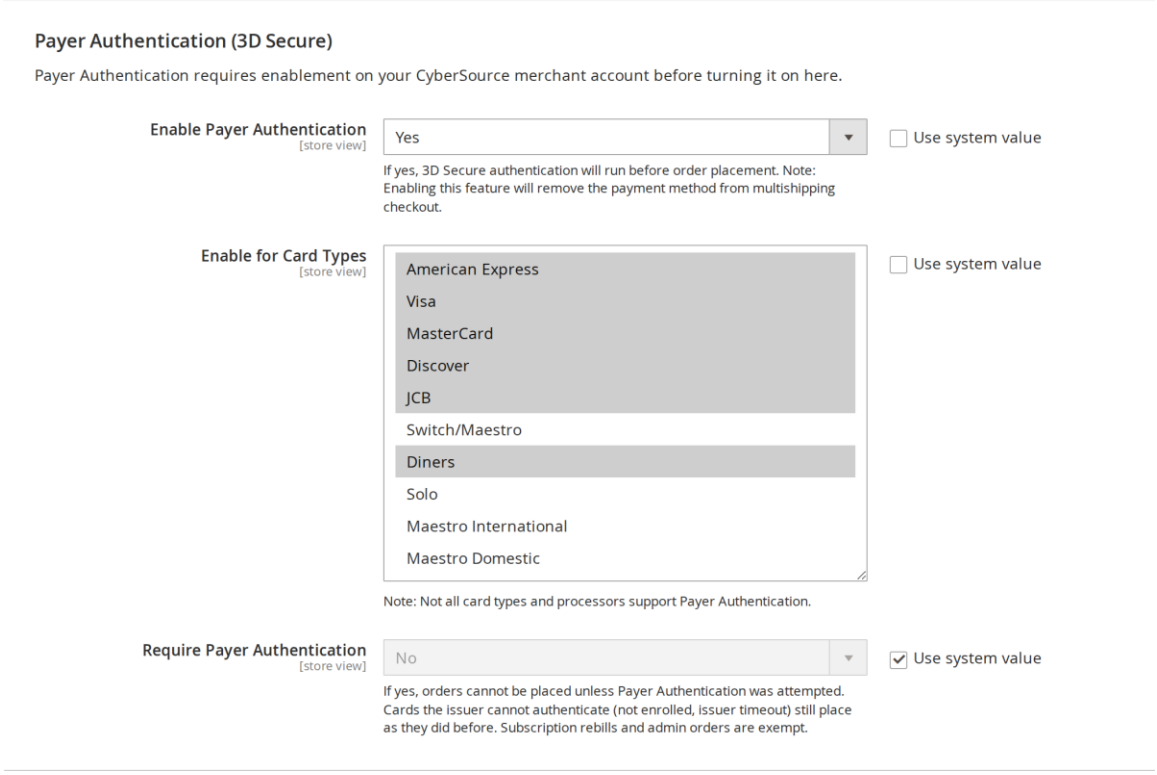
All credit card forms use the CyberSource Unified Checkout solution (UC.js), authenticated with the REST API keys above. Digital wallets must be enabled on your CyberSource merchant account before you turn them on here.



- **Allowed Payment Types**: the methods offered inside the Unified Checkout form: Card Entry, Apple Pay, Google Pay, Click to Pay, and Paze. Leave only Card Entry selected unless the wallet is enabled on your merchant account. At least one type is required.
- **Allowed Card Networks**: the card networks the form will accept (Visa, Mastercard, American Express, Discover, Diners Club, JCB, Maestro, China UnionPay, Cartes Bancaires). Select only the networks your merchant account supports.
- **Show CyberSource Logo**: if Yes, a CyberSource logo is shown on the payment form.
- **Show Payment Review**: off by default. If Yes, CyberSource shows its own review screen after card entry before handing control back to checkout, and the final button reads ‘Pay’ or ‘Confirm and Continue’ rather than ‘Continue’.

Payer Authentication (3D Secure)

Payer Authentication (3D Secure 2) must be enabled on your CyberSource merchant account before you turn it on here. Version 4.0 runs 3D Secure natively on the REST API — no CardinalCommerce credentials are required.

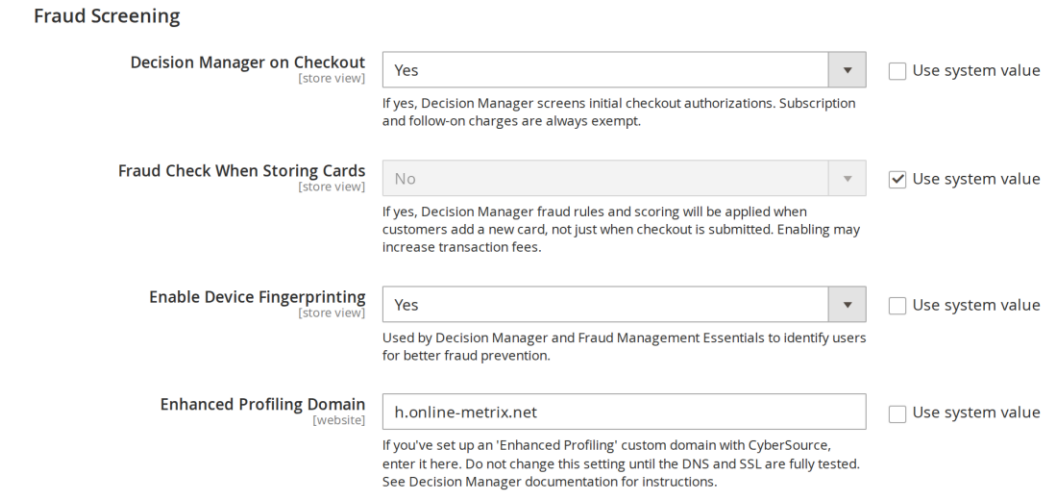


- **Enable Payer Authentication**: if Yes, 3D Secure runs before order placement. Note: enabling this removes the payment method from multishipping checkout.
- **Enable for Card Types**: the card types that will attempt 3D Secure.

Important: an empty selection disables Payer Authentication for every card even when Enable Payer Authentication is Yes — every charge is then out of scope. The 4.0 default is American Express, Visa, Mastercard, Discover, JCB, and Diners Club; EEA merchants accepting Maestro should add it.

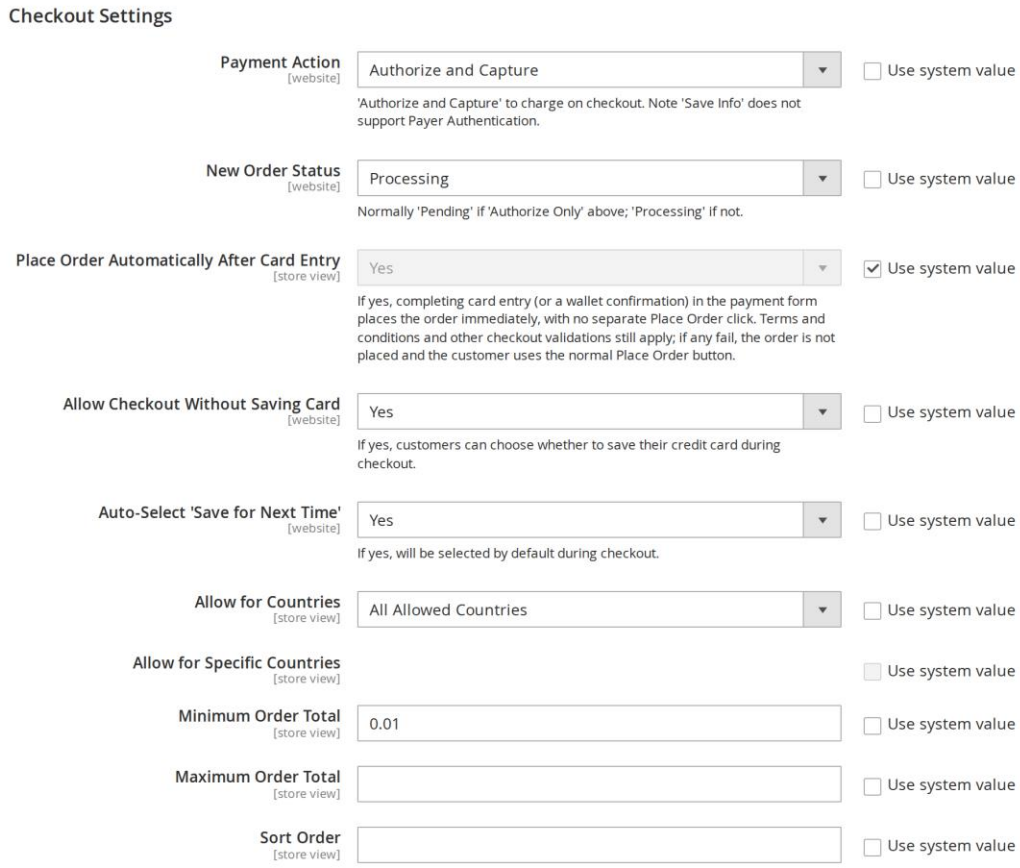
- **Require Payer Authentication**: off by default. If Yes, an order cannot be placed unless Payer Authentication was attempted. It requires the authentication to be attempted, not to succeed: a card the issuer cannot authenticate (not enrolled, issuer timeout, directory-server error) still places without a liability shift. Subscription rebills and admin orders are exempt. A related origins field for headless storefronts, Payer Auth Return URL Origins, appears under Advanced Settings.

Fraud Screening



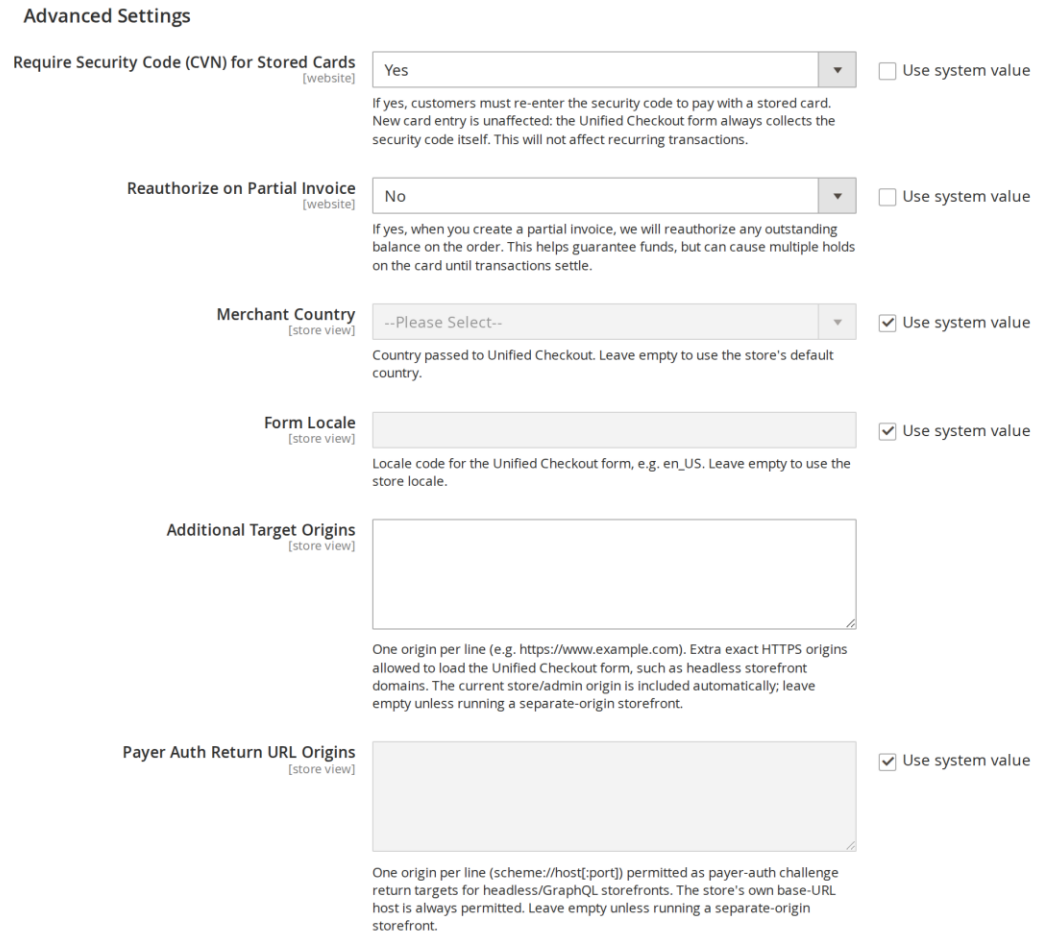
- **Decision Manager on Checkout**: if Yes, Decision Manager screens the initial checkout authorization. Subscription and follow-on charges are always exempt.
- **Fraud Check When Storing Cards**: if Yes, Decision Manager rules and scoring are also applied to the $0 authorization used when a customer adds a new card, not just at checkout. Off by default; enabling may cause additional transaction fees.
- **Enable Device Fingerprinting**: loads CyberSource's device-fingerprinting script (from h.online-metrix.net by default) so Decision Manager and Fraud Management Essentials can identify users. May add third-party cookies to checkout.
- **Enhanced Profiling Domain**: if you have configured an 'Enhanced Profiling' custom domain with CyberSource, enter it here so the fingerprint script loads from your domain. Do not change this until the DNS and SSL are fully tested.

Checkout Settings



- **Payment Action**: when the order is charged:
    - Save Info (do not authorize): stores the card on checkout without authorizing. Note Save Info does not support Payer Authentication.
    - Authorize: authorizes the order amount on checkout for later manual invoicing/capture.
    - Authorize and Capture: captures funds immediately when the order is placed.
- **New Order Status**: the initial status for CyberSource orders — normally ‘Pending’ for Authorize Only, ‘Processing’ otherwise.
- **Place Order Automatically After Card Entry**: new in 4.0. If Yes, completing card entry (or a wallet confirmation) in the payment form places the order immediately, with no separate Place Order click. Terms and conditions and other checkout validations still apply; if any fail, the order is not placed and the customer uses the normal Place Order button.
- **Allow Checkout Without Saving Card**: if Yes, customers get a ‘Save for next time’ choice. If No, logged-in customers see a stored-securely message and guests are never offered card storage.
- **Auto-Select ‘Save for Next Time’**: if Yes, the save checkbox is selected by default.
- **Allow for Countries**: limit which billing countries may use this payment method.
- **Minimum / Maximum Order Total**: restrict the order value range for which the method is offered.
- **Sort Order**: the position of this method relative to other payment options on checkout.

Advanced Settings



- **Require Security Code (CVN) for Stored Cards**: if Yes, customers must re-enter the security code to pay with a stored card. New card entry is unaffected — the Unified Checkout form always collects the security code itself. Does not affect recurring transactions.
- **Reauthorize on Partial Invoice**: if Yes, creating a partial invoice reauthorizes any outstanding balance on the order. This helps guarantee funds but can place multiple holds on the card until transactions settle.
- **Merchant Country**: new in 4.0. Country passed to Unified Checkout. Leave empty to use the store's default country.
- **Form Locale**: new in 4.0. Locale code for the Unified Checkout form, e.g. en_US. Leave empty to use the store locale.
- **Additional Target Origins**: new in 4.0. One HTTPS origin per line permitted to load the Unified Checkout form, such as headless storefront domains. The current store/admin origin is included automatically; leave empty unless running a separate-origin storefront.
- **Payer Auth Return URL Origins**: new in 4.0. One origin per line permitted as a Payer Authentication challenge return target for headless/GraphQL storefronts. The store's own base-URL host is always permitted. Leave empty unless running a separate-origin storefront.

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 stored-card functionality and everything related to it. A few things are worth noting.

User Experience

Version 4.0 replaces the old Secure Acceptance iframe with the CyberSource Unified Checkout drop-in payment form. All card data is still entered directly into CyberSource-hosted fields, so the flow differs slightly from a typical payment method:

- The customer enters and confirms their billing address first, on the standard Magento checkout. This may default to "My billing and shipping address are the same," in which case the payment form loads immediately.
- The Unified Checkout form then presents the enabled payment types — "Pay with card" plus any enabled wallets (Google Pay, Apple Pay, Click to Pay, Paze). Choosing "Pay with card" reveals the embedded card-entry fields.
- Completing card entry tokenizes the card. With Place Order Automatically After Card Entry set to Yes (the 4.0 default), and consent already given, the order is placed at that moment and no separate Place Order click is needed. With it set to No, the tokenized card is shown as selected and the customer clicks Place Order.
- If Require Security Code (CVN) for Stored Cards is on, paying with a previously stored card re-prompts for the security code before the order can be placed. Newly entered cards always collect it within the form.
- If Payer Authentication is enabled, the customer may be shown a 3D Secure challenge overlay before the order completes; see below.

Card management within a customer's account (frontend or admin) follows the same pattern: the billing address is entered and confirmed before the Unified Checkout form is shown to add or update a card. When editing a stored card, the expiration date can be updated without re-entering the full card number.

Security

The Unified Checkout form is always used; there is no option to collect raw card details on your own server. All card data (number, expiration, security code) is entered into CyberSource-hosted fields and transmitted directly to CyberSource, which returns a short-lived transient token. At no time does a card number touch your server for this payment method. This keeps the best possible mix of user experience and PCI scope.

Some non-confidential card metadata is kept in your database — card type, last four digits, expiration date, and CyberSource identifiers, among other fields. All of it is available via the CyberSource API, and none of it is considered Confidential Cardholder Data under PCI.

If you intend to collect payments from other sources using the Magento API, you must tokenize the card through CyberSource (via the Unified Checkout transient token, or another CyberSource API) and store the resulting token as a TokenBase Card via the REST or GraphQL API. See *Technical / Integration Details*. This payment method will not accept a raw card number under any circumstances.

3D Secure / Payer Authentication

Version 4.0 runs 3D Secure 2 natively on the CyberSource REST API. The CardinalCommerce Songbird integration used in 3.x is gone, and no CardinalCommerce credentials are needed — but Payer Authentication must be enabled on your CyberSource merchant account.

3D Secure applies to standard frontend checkout only. It is not compatible with multishipping checkout: enabling it removes the CyberSource method from multishipping. It also does not apply to admin or internal transactions — there is no 3D Secure coverage for admin-created orders, which run with a 'Mail or Telephone Order' (MOTO) commerce indicator and are exempt. Subscription rebills are likewise exempt.

When a challenge is required, the customer is shown their issuer's authentication overlay on top of the checkout page. The exact prompt varies by card and enrollment. On success, checkout proceeds automatically. A card that the issuer cannot authenticate (not enrolled, issuer timeout, directory-server error) still places without a liability shift, as it did in 3.x.

A Payer Authentication result is scoped to the card, not the cart. A card that fails or abandons 3D Secure stays blocked until it authenticates; switching to a different card clears the block on the same cart. Re-entering the same card number counts as a different card (it authenticates from scratch), while a stored card stays blocked until it passes. A blocked card reports "Your payment could not be verified. Please re-enter your payment information and try again." and the payment form is reset, rather than silently re-running a verification it can never pass.

Require Payer Authentication (off by default) refuses any order placed without attempting Payer Authentication, which prevents any ability to bypass it via REST/GraphQL API. It requires the attempt, not success; subscription rebills and admin orders are exempt.

Card Storage

Credit cards are always stored in CyberSource Token Management Service to enable advanced functionality, even when the customer chooses not to save. If the customer opts out or is a guest, the card is not shown for reuse within Magento and is automatically purged from all systems 120 days after its last use. This keeps the card available for edits, captures, and refunds while respecting the customer's wishes.

There is no automatic duplicate prevention: entering the same card three times stores it in CyberSource three times. CyberSource imposes no limit on the number of stored cards/tokens.

Account Updater

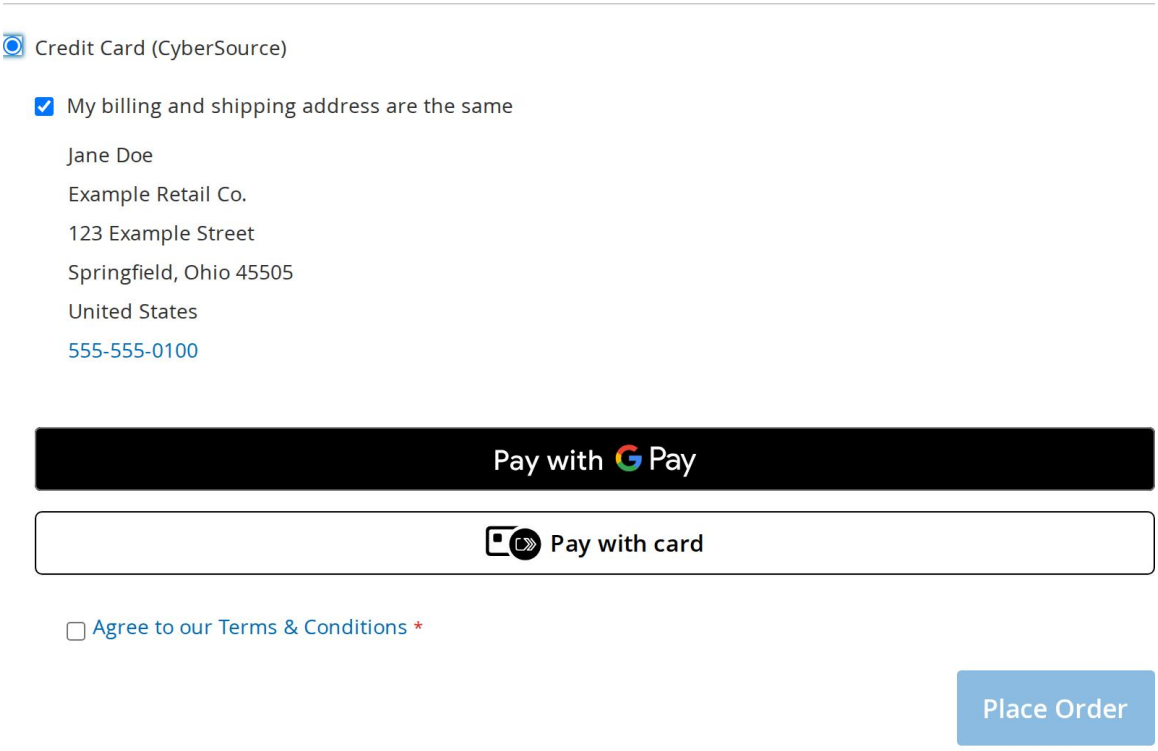
Account Updater is supported and automatically active if REST API credentials are configured and Account Updater is enabled on your CyberSource account. There is a charge for CyberSource's Account Updater service. When enabled, your stored cards in Magento automatically update to reflect changes in expiration dates, card numbers, and closed accounts.

Usage

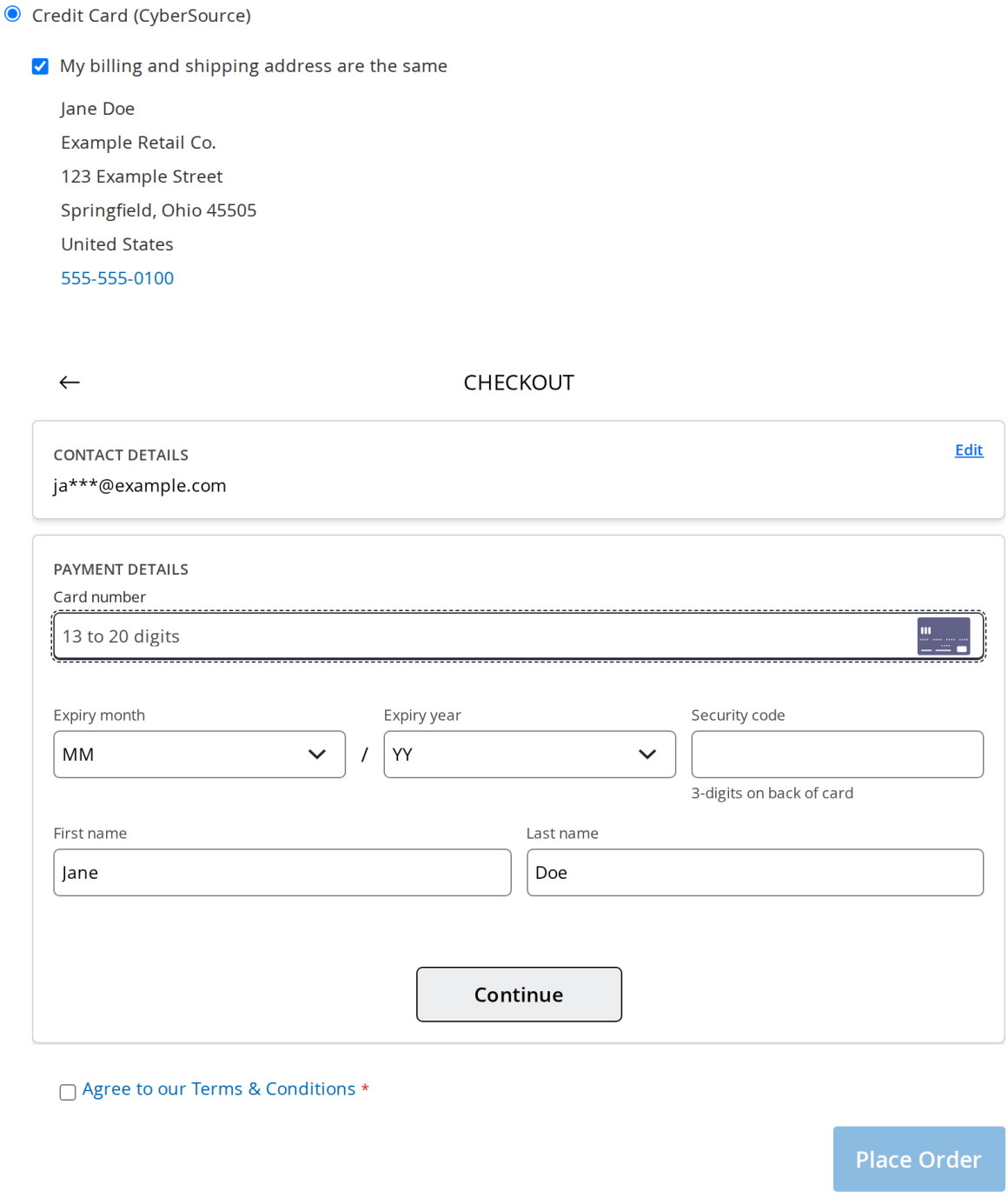
There isn't much to using this extension in practice: it's a standard Magento payment method, and the interfaces should be self-explanatory. Here's a rundown of what you get.

Checkout Payment Form

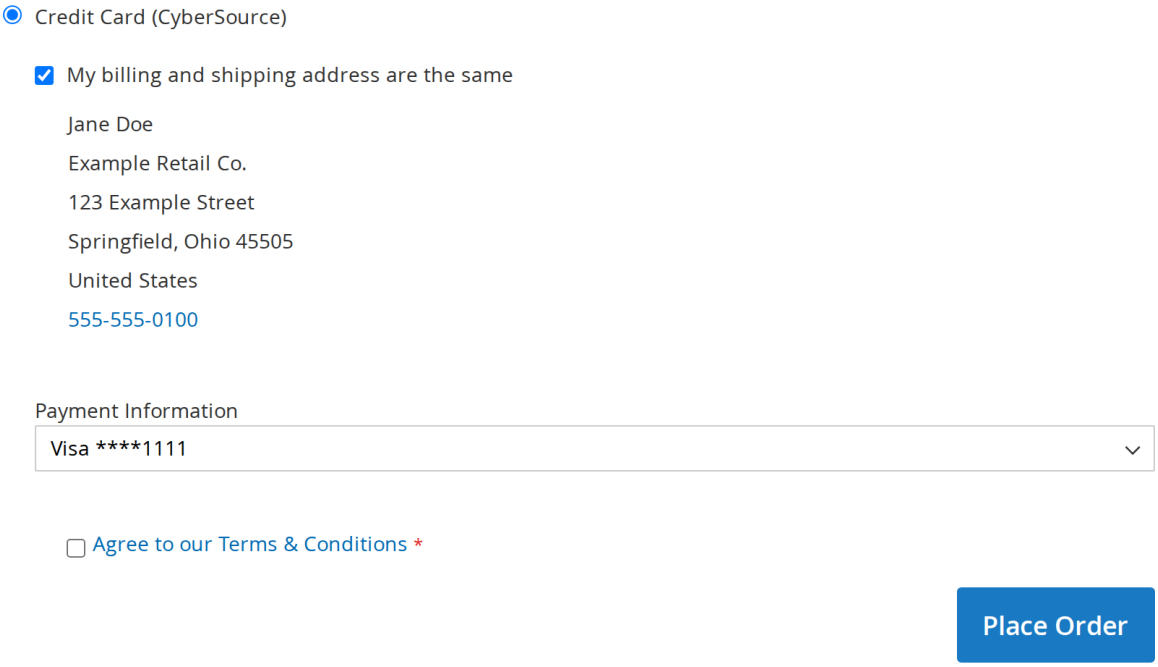
On checkout, the customer confirms their billing address and is then shown the CyberSource Unified Checkout form. It lists the enabled payment options — “Pay with card” plus any enabled wallets:



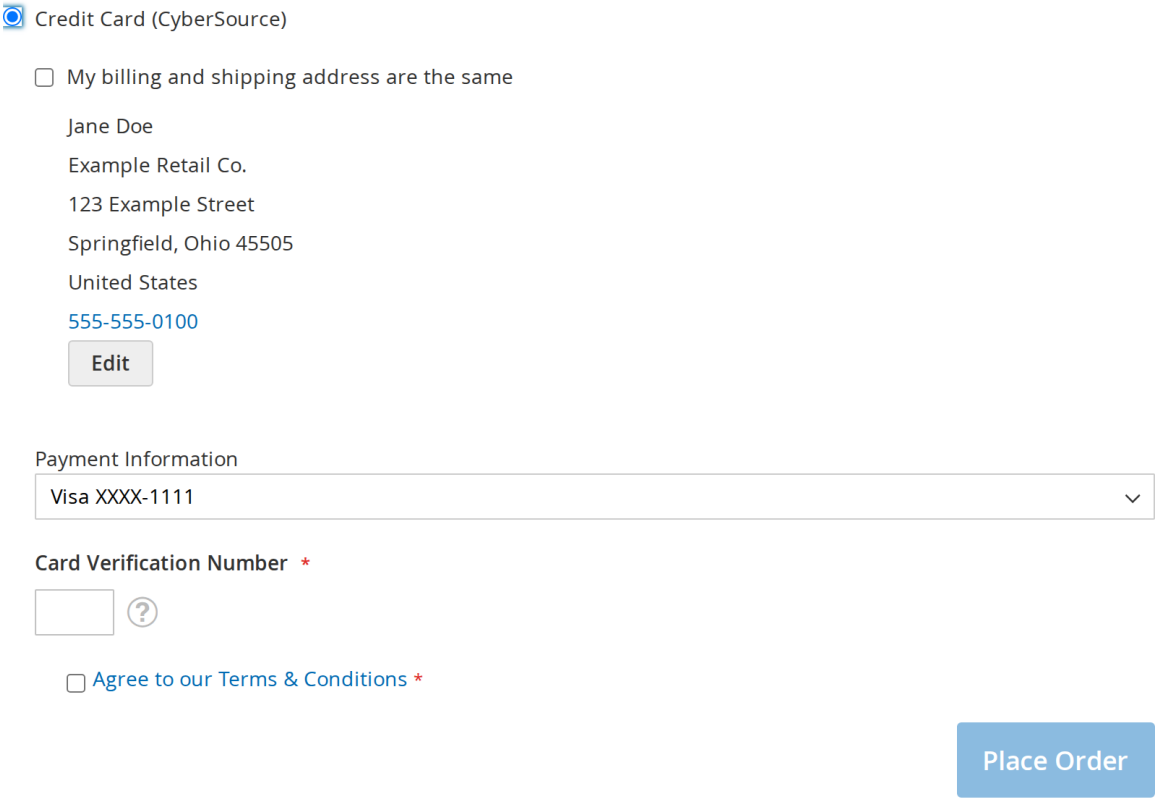
Choosing “Pay with card” reveals the embedded card-entry fields. All card data is entered directly into CyberSource-hosted fields; any format errors are shown inline as the customer types.



With Place Order Automatically After Card Entry set to Yes (the 4.0 default), completing the card form places the order immediately. With it set to No, the tokenized card is shown ready and the customer clicks Place Order:



Logged-in customers with a stored card see their most recent card selected by default. If Require Security Code (CVN) for Stored Cards is enabled, they re-enter the security code before placing the order:



When card saving is offered (Allow Checkout Without Saving Card), a “Save for next time” option appears with the card form:

Credit Card (CyberSource)

My billing and shipping address are the same

Jane Doe
Example Retail Co.
123 Example Street
Springfield, Ohio 45505
United States
555-555-0100

← CHECKOUT

CONTACT DETAILS Edit
jan***@example.com

PAYMENT DETAILS
Card number
13 to 20 digits

Expiry month: MM / Expiry year: YY
Security code
3-digits on back of card

First name: Jane
Last name: Doe

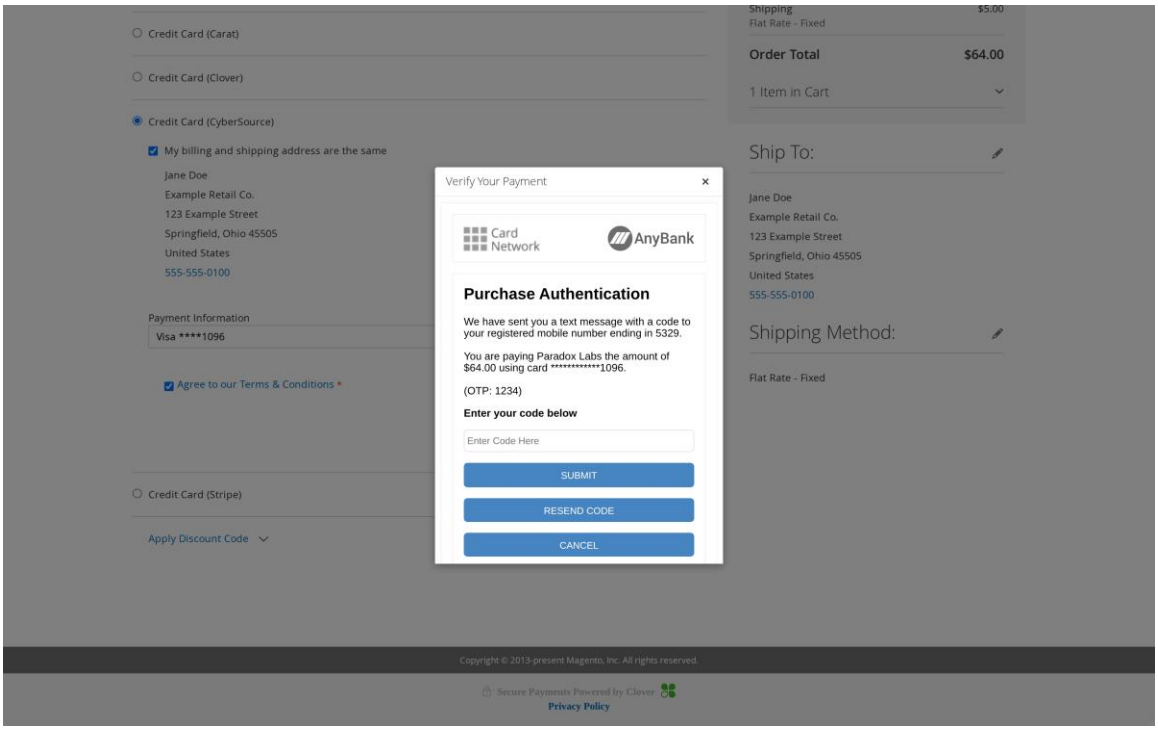
Continue

Save for next time

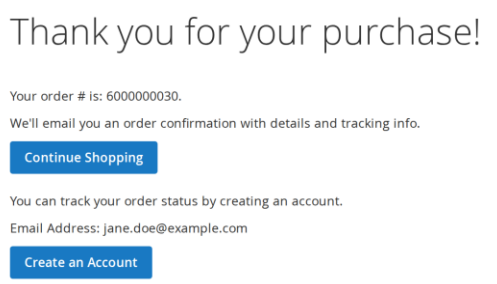
Place Order

Payer Authentication (3D Secure)

If Payer Authentication is enabled, some cards require a 3D Secure challenge before the order is placed. The customer is shown their issuer's authentication overlay on top of the checkout page; its exact appearance varies by card and issuer:



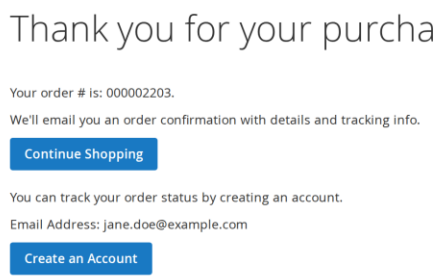
On a successful challenge, checkout completes automatically and the order is placed:



A card the issuer cannot authenticate (not enrolled, issuer timeout, directory-server error) still places without a liability shift. A card that fails or abandons authentication is blocked and the customer sees *"Your payment could not be verified. Please re-enter your payment information and try again."* with the form reset; the block is scoped to that card, so switching to another card lets checkout proceed. Payer Authentication does not apply to admin-created orders — there is no 3D Secure coverage in the admin, which runs with a Mail/Telephone Order (MOTO) indicator and is exempt.

Order status page

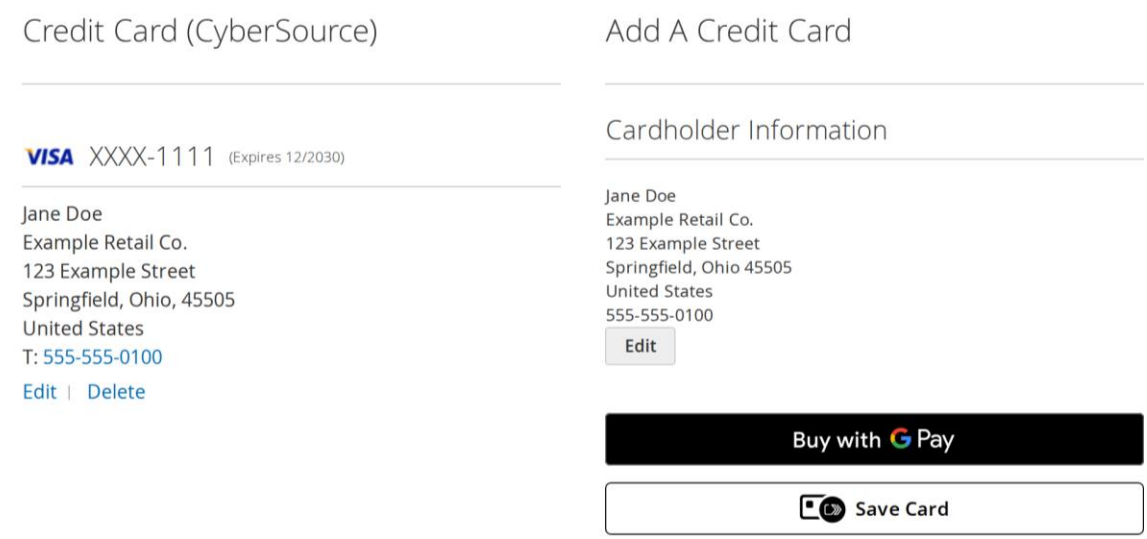
After placing an order, the customer sees the standard Magento order success and status pages:



Customer 'My Payment Data' account area

The My Payment Data section lets customers view their stored cards and add, edit, or delete them.

As on checkout, the billing address must be entered and confirmed before the payment form is shown, so adding or editing a card is a two-step process. After confirming the address, the Unified Checkout form mounts its own card and wallet controls:



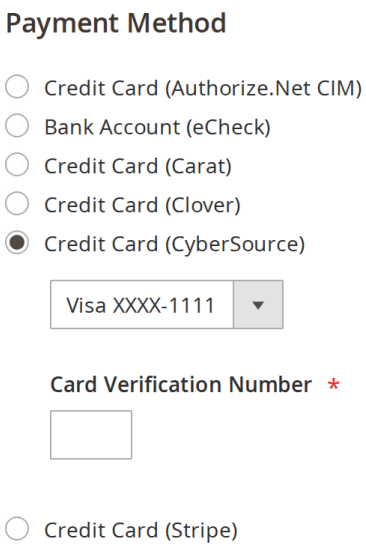
When editing a card you can change the address without touching the card details, or update the expiration date without re-entering the full card number. Any change to the card details re-collects the security code.

Cards associated with an open (uncaptured) order cannot be edited or deleted; they display a 'Card In Use' message in place of the buttons until all orders paid by the card are complete.

To prevent abuse, the Payment Data section is only available after the customer has placed a successful order; before then they are redirected to the Account Dashboard with *"My Payment Data will be available after you've placed an order."* If a customer hits errors saving a card five times, they are blocked for 24 hours with "My Payment Data is currently unavailable. Please try again later." Both behaviors can be adjusted or disabled — contact us if needed.

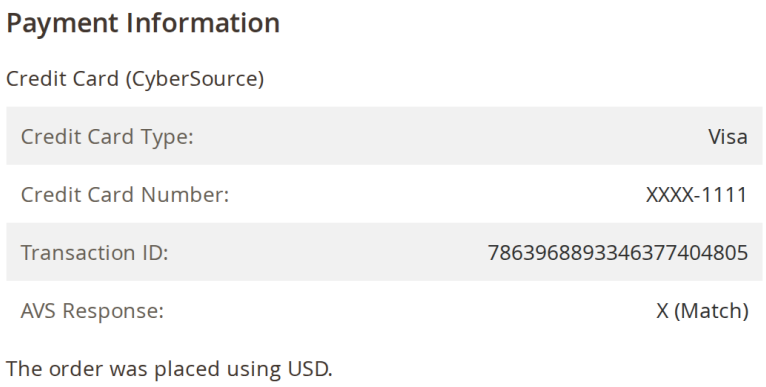
Admin order form

The admin Create New Order form offers the same options and behavior as frontend checkout, including stored-card reuse:



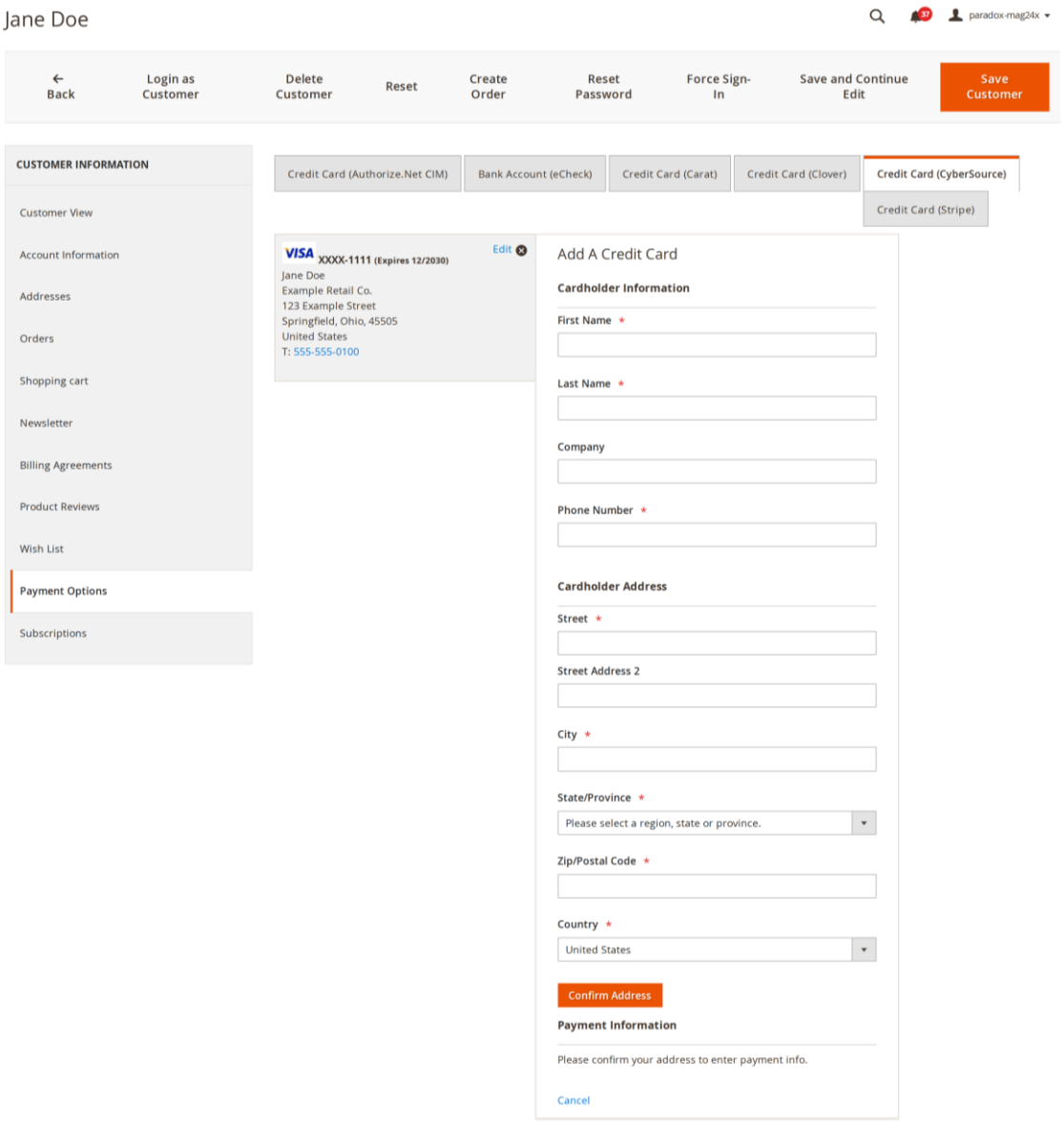
Admin order status page

After an order is placed, the admin shows extended payment information — transaction ID, validation responses, and fraud score (if using Decision Manager). None of this is visible to the customer:



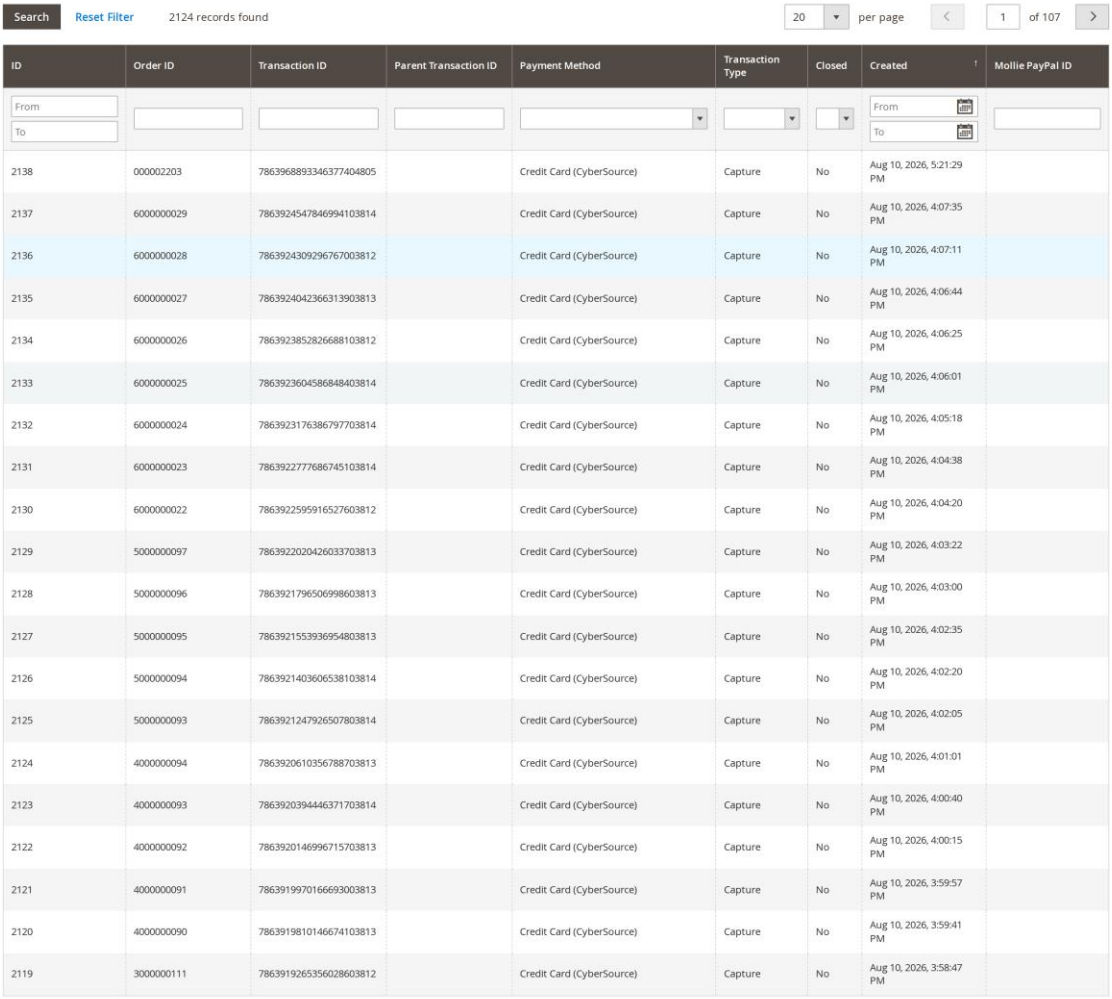
Admin customer ‘Payment Data’ account area

Viewing a customer, an added ‘Payment Data’ tab offers the same view/add/edit/delete functionality as the frontend section, for managing your customers’ stored CyberSource cards:



Admin transaction info

Viewing an order, the ‘Transactions’ tab shows full transaction info:



Click into a transaction to see the raw data returned by the CyberSource REST API — including tokenInformation, processorInformation, and, for 3D Secure orders, consumerAuthenticationInformation. This same data is accessible from the transaction record in code.

← Back Fetch

Transaction Data

Transaction ID	786396889334637740 4805
Parent Transaction ID	N/A
Order ID	000002203
Transaction Type	capture
Is Closed	No
Created At	Aug 10, 2026, 5:21:29 PM

Child Transactions

0 records found

ID	Order ID	Transaction ID	Payment Method	Transaction Type	Closed	Created	Mollie PayPal ID
We couldn't find any records.							

Transaction Details

Key	Value
_links.void.method	POST
_links.void.href	/pts/v2/payments/7863968893346377404805/voids
_links.self.method	GET
_links.self.href	/pts/v2/payments/7863968893346377404805
clientReferenceInformation.code	000002203
clientReferenceInformation.partner.solutionId	DEQXVEEG
consumerAuthenticationInformation.token	AxJ/ws Tqk5922fYJGFABEg3atWrlqyb16c6a1jzXle/d2eMQgKe/d2eMQ6QNyAMJ0MmkmXoxXz9FBgTk6pOfdtn2IyRhQA4gdl
id	7863968893346377404805
orderInformation.amountDetails.totalAmount	64.00
orderInformation.amountDetails.authorizedAmount	64.00
orderInformation.amountDetails.currency	USD
paymentAccountInformation.card.type	001
paymentInformation.tokenizedCard.type	001
paymentInformation.card.type	001
pointOfSaleInformation.terminalId	111111
processorInformation.paymentAccountReferenceNumber	7D8F0BA0099CD03C00096520312F0
processorInformation.approvalCode	888888
processorInformation.networkTransactionId	123456789619999
processorInformation.transactionId	123456789619999
processorInformation.responseCode	100
processorInformation.avs.code	X
processorInformation.avs.codeRaw	I1
reconciliationId	75559527WSNM5GLT
status	AUTHORIZED
submitTimeUtc	2026-08-10T21:21:29Z
tokenInformation.instrumentIdentifierNew	0
tokenInformation.instrumentIdentifier.state	ACTIVE
tokenInformation.instrumentIdentifier.id	7010000000082721111
tokenInformation.thirdPartyToken.id	58B8EA6D89BB1902E063AF598E0AA18E
tokenInformation.paymentInstrument.id	58B8EA6D89BB1902E063AF598E0AA18E
transaction_id	7863968893346377404805
response_code	100
response_reason_code	100
response_reason_text	AUTHORIZED
auth_code	888888
ccAuthReply.authorizationCode	888888
ccAuthReply.avsCode	X
token_information.paymentInstrument	58B8EA6D89BB1902E063AF598E0AA18E
token_information.instrumentIdentifier	7010000000082721111
uc_token_missing	0
card_information.cc_type	VI
card_information.cc_last4	1111
card_information.cc_exp_month	12
card_information.cc_exp_year	2030
consumer_authentication.token	AxJ/ws Tqk5922fYJGFABEg3atWrlqyb16c6a1jzXle/d2eMQgKe/d2eMQ6QNyAMJ0MmkmXoxXz9FBgTk6pOfdtn2IyRhQA4gdl
is_fraud	0
is_error	0

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 collects all card data in CyberSource-hosted Unified Checkout fields for every credit card form, and does not support collecting card data by any other means. Because a card number never touches your server, this is intended to support the reduced-scope SAQ A eligibility (verify your specific obligations with your acquirer).

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 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`. In 4.0 it reaches CyberSource entirely over the REST API. `\ParadoxLabs\CyberSource\Model\Gateway` (extending TokenBase's `AbstractGateway`) routes the gateway operations to REST service classes under `\ParadoxLabs\CyberSource\Model\Service`, notably `Service\Rest` (the signed HTTP client) and the `Service\UnifiedCheckout` and `Service\PayerAuth` services. Card metadata is stored in instances of `\ParadoxLabs\CyberSource\Model\Card`. Each of these extends an equivalent abstract class in TokenBase and implements only the CyberSource-specific details.

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 TokenBase Card with a valid CyberSource card token (obtained from a Unified Checkout transient token, or another CyberSource tokenization API) plus the associated card and address data. To place an order with a stored card, pass that card's hash as additional_data → card_id, along with cc_cid for the security code (if required) and save (true or false) for whether it should be retained.

Raw credit card numbers (cc_number) are never accepted. You must tokenize the card with CyberSource first — the simplest path is the Unified Checkout transient token described below — then place the order with that transient token or a stored card hash. The token flow is detailed below in 'How-To: API Checkout Flow'.

To run Payer Authentication (3D Secure) outside the built-in checkout, use the CyberSource Payer Authentication operations — cyberSourcePayerAuthSetup, cyberSourcePayerAuthAuthenticate, and cyberSourcePayerAuthFinalize — exposed as GraphQL mutations and as REST endpoints. The authentication result is bound to the cart and card server-side, so a later order placement is allowed automatically; the client sends no authentication payload. The full sequence is in 'How-To: API 3D Secure Flow'. (The 3.x CardinalCommerce Cardinal Cruise integration and its response_jwt field are gone.)

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

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.

Raw credit card numbers (cc_number) are never accepted. You must tokenize the card with CyberSource first — the simplest path is the Unified Checkout transient token described below — then place the order with that transient token or a stored card hash. The token flow is detailed below in 'How-To: API Checkout Flow'.

To run Payer Authentication (3D Secure) outside the built-in checkout, use the CyberSource Payer Authentication operations — cyberSourcePayerAuthSetup, cyberSourcePayerAuthAuthenticate, and cyberSourcePayerAuthFinalize — exposed as GraphQL mutations and as REST endpoints. The authentication result is bound to the cart and card server-side, so a later order placement is allowed automatically; the client sends no authentication payload. The full sequence is in 'How-To: API 3D Secure Flow'. (The 3.x CardinalCommerce Cardinal Cruise integration and its response_jwt field are gone.)

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.

cyberSourceUnifiedCheckoutCaptureContext(input: TokenBaseCyberSourceUnifiedCheckoutInput!): TokenBaseCyberSourceUnifiedCheckoutCaptureContext
Get a CyberSource Unified Checkout capture-context JWT for collecting payment data. Provide the cart ID (and guestEmail for a guest cart). The returned JWT is used to initialize the Unified Checkout form on the client.

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.

cyberSourcePayerAuthSetup(input: TokenBaseCyberSourcePayerAuthSetupInput!): TokenBaseCyberSourcePayerAuthSetupResult
Start CyberSource Payer Authentication (3D Secure) for a cart and return the device-data-collection parameters. Provide exactly one of transientToken (a new card) or cardHash (a stored card).

cyberSourcePayerAuthAuthenticate(input: TokenBaseCyberSourcePayerAuthAuthenticateInput!): TokenBaseCyberSourcePayerAuthResult
Run Payer Authentication for a cart after device data collection, passing client-collected browserInfo. Returns a status of success, failed, challenge, or skipped.

cyberSourcePayerAuthFinalize(input: TokenBaseCyberSourcePayerAuthFinalizeInput!): TokenBaseCyberSourcePayerAuthResult
Return the final Payer Authentication outcome for a cart after an issuer challenge.

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?
        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
        fingerprintUrl: String                          Script URL for fingerprinting, if enabled
}
```

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                                       Gateway card token (unused for CyberSource; use
transient_token)
        save: Boolean                                       Save the card for later use? (checkout only)
        card_id: String                                     Card identifier hash to use (checkout only)
        transient_token: String                             Unified Checkout transient-token JWT (new card;
                                                            provide one of transient_token or card_id)
}
```

TokenBaseCyberSourceUnifiedCheckoutInput

Input for requesting a Unified Checkout capture context.

```
input TokenBaseCyberSourceUnifiedCheckoutInput {
  cartId: String                        Masked cart ID (for checkout)
  guestEmail: String                    Guest email, when the cart has no logged-in customer
  billingAddress: CustomerAddressInput  Billing address, when supplied outside the quote
}
```

TokenBaseCyberSourceUnifiedCheckoutCaptureContext

The Unified Checkout capture context.

```
type TokenBaseCyberSourceUnifiedCheckoutCaptureContext {
  captureContext: String        Capture-context JWT used to initialize the Unified Checkout form
}
```

TokenBaseCyberSourcePayerAuthSetupInput

Input for starting Payer Authentication on a cart. Provide exactly one of transientToken or cardHash.

```
input TokenBaseCyberSourcePayerAuthSetupInput {
  cartId: String                Masked cart ID (required)
```

```
  guestEmail: String            Guest email, when the cart has none
  transientToken: String        Unified Checkout transient-token JWT for a new card
  cardHash: String              Public hash of a stored card
}
```

TokenBaseCyberSourcePayerAuthSetupResult

The result of starting Payer Authentication.

```
type TokenBaseCyberSourcePayerAuthSetupResult {
  skipped: Boolean!                 True when 3D Secure does not apply; place the order without it
  accessToken: String               JWT to POST to the device-data-collection URL
  deviceDataCollectionUrl: String   URL to POST the access token to, in a hidden iframe
}
```

TokenBaseCyberSourcePayerAuthAuthenticateInput

Input for running Payer Authentication on a cart, after device data collection.

```
input TokenBaseCyberSourcePayerAuthAuthenticateInput {
  cartId: String                Masked cart ID (required)
  guestEmail: String            Guest email, when the cart has none
  returnUrl: String             Absolute https URL the challenge returns to (may be a
                                separate storefront origin; defaults to a store-hosted URL)
  browserInfo: TokenBaseCyberSourcePayerAuthBrowserInfoInput   Client browser data (required)
}
```

TokenBaseCyberSourcePayerAuthBrowserInfoInput

Client-collected browser data required by the card networks.

```
input TokenBaseCyberSourcePayerAuthBrowserInfoInput {
  language: String!             navigator.language
  javaEnabled: Boolean!         navigator.javaEnabled()
  javaScriptEnabled: Boolean!   True for browser clients
  colorDepth: Int!              screen.colorDepth (bits)
  screenHeight: Int!            screen.height (px)
  screenWidth: Int!             screen.width (px)
  timeDifference: Int!          Date.getTimezoneOffset() (minutes)
}
```

TokenBaseCyberSourcePayerAuthFinalizeInput

Input for retrieving the final Payer Authentication outcome for a cart.

```
input TokenBaseCyberSourcePayerAuthFinalizeInput {
  cartId: String                Masked cart ID (required)
  guestEmail: String            Guest email, when the cart has none
}
```

TokenBaseCyberSourcePayerAuthResult

The outcome of a Payer Authentication attempt. The authentication data itself stays server-side.

```
type TokenBaseCyberSourcePayerAuthResult {
  status: String!               success, failed, challenge, or skipped
  acsUrl: String                Issuer ACS URL (informational). Only set when status is challenge
  pareq: String                 Challenge request payload (informational). Only when challenge
  stepUpUrl: String             URL to POST the access token to, in an iframe, to run the challenge
  accessToken: String           JWT to POST to the step-up URL. Only set when status is challenge
}
```

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, hasVerification, cvvImageUrl,
    availableTypes {key, value}, months {key, value}, years {key, value},
    fingerprintUrl
  }
}
```

Example response:

```
{
  "data": {
    "tokenBaseCheckoutConfig": {
      "method": "paradoxlabs_cybersource",
      "useVault": true,
      "canSaveCard": true,
      "forceSaveCard": false,
      "defaultSaveCard": true,
      "logoImage": "false",
      "requireCcv": false,
      "hasVerification": true,
      "cvvImageUrl": "https://example.com/.../cvv.png",
      "availableTypes": [ { "key": "VI", "value": "Visa" }, ... ],
      "months": [ { "key": "1", "value": "01 - January" }, ... ],
      "years": [ { "key": "2026", "value": "2026" }, ... ],
      "fingerprintUrl": "https://h.online-metrix.net/fp/tags.js?..."
    }
  }
}
```

The Unified Checkout capture context and Payer Authentication data are fetched with their own queries/mutations, below. fingerprintUrl is the optional Decision Manager device-fingerprint script.

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 Unified Checkout capture context

Example request:

```
query {
  cyberSourceUnifiedCheckoutCaptureContext(input: {
    cartId: "kDyOEKkJmIOxa6H3ceus6MjMaSF9lao1"
  }) {
    captureContext
  }
}
```

Example response (the capture context is a JWT):

```
{
  "data": {
    "cyberSourceUnifiedCheckoutCaptureContext": {
      "captureContext": "eyJraWQiOiI...<JWT>..."
    }
  }
}
```

Run Payer Authentication (3D Secure)

Three mutations run in sequence: setup, authenticate, then (only after a challenge) finalize. Provide exactly one of transientToken or cardHash on setup.

Setup — returns the device-data-collection parameters:

```
mutation {
  cyberSourcePayerAuthSetup(input: {
    cartId: "kDyOEKkJmIOxa6H3ceus6MjMaSF9lao1"
    transientToken: "eyJ...<from Unified Checkout>"
  }) {
```

```
    skipped
    accessToken
    deviceDataCollectionUrl
  }
}
```

Authenticate — after device data collection; returns success, failed, skipped, or challenge:

```
mutation {
  cyberSourcePayerAuthAuthenticate(input: {
    cartId: "kDyOEKkJmIOxa6H3ceus6MjMaSF9lao1"
    returnUrl: "https://www.example.com/checkout"
    browserInfo: {
      language: "en-US", javaEnabled: false, javaScriptEnabled: true,
      colorDepth: 24, screenHeight: 1080, screenWidth: 1920, timeDifference: 300
    }
  }) {
    status
    stepUpUrl
    accessToken
  }
}
```

Finalize — only when authenticate returned challenge, after the challenge completes:

```
mutation {
  cyberSourcePayerAuthFinalize(input: {
    cartId: "kDyOEKkJmIOxa6H3ceus6MjMaSF9lao1"
  }) {
    status
  }
}
```

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,
        }
      }
    }
  ) {
    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

In 4.0 the card form is the CyberSource Unified Checkout drop-in, and all processing runs on the REST API. A headless or custom integration replicates four steps: obtain a capture context, mount the drop-in to collect a transient token, optionally run Payer Authentication (next section), then place the order with that transient token or a stored card hash.

Step 1: Fetch a capture context

For GraphQL, query `cyberSourceUnifiedCheckoutCaptureContext` (see the example above). Standard checkout uses the frontend AJAX controller `pdl_cybs/unifiedCheckout/captureContext` (admin: the equivalent adminhtml controller); there is no webapi/REST route for the capture context. Provide the cart ID, guestEmail for guests, and billingAddress if supplying an address outside the quote.

The response is a capture-context JWT. It encodes the allowed payment types and card networks, the permitted target origins, and the URL (with SRI hash) of the CyberSource Unified Checkout client library — you do not hardcode that library URL.

Step 2: Mount the Unified Checkout drop-in

Decode the capture-context JWT and load the client library named in its clientLibrary field, honoring clientLibraryIntegrity to ensure subresource integrity. Initialize the drop-in with the capture context and show it in your containers. When the customer completes card entry (or a wallet), the drop-in resolves with a transient-token JWT:

```
var uc = await Accept(captureContext);
var up = await uc.unifiedPayments(false);
var transientToken = await up.show({
    containers: { paymentSelection: '#pay-select', paymentScreen: '#pay-screen' }
});
```

The transient token’s decoded payload carries display metadata (card type, last four, BIN prefix, expiry) but no full card number. It is single-use and is consumed when the order is authorized.

Step 3: (Optional) Run Payer Authentication

If Payer Authentication is enabled, run the setup / authenticate / finalize sequence described in 'How-To: API 3D Secure Flow' before placing the order. If setup returns `skipped: true`, 3D Secure does not apply and you can place the order directly.

Step 4: Place the order

Submit the order with the transient token as `tokenbase_data.transient_token` (new card) or `card_id` plus `cc_cid` (stored card), along with `save`. The stored Payer Authentication result, if any, is validated and consumed server-side; the client sends no authentication payload.

GraphQL example:

```
mutation {
  setPaymentMethodOnCart(input: {
    cart_id: "kDyOEKkJmIOxa6H3ceus6MjMaSF9lao1"
    payment_method: {
      code: "paradoxlabs_cybersource",
      tokenbase_data: {
        transient_token: "eyJ...<from Unified Checkout>",
        save: true
      }
    }
  }) { cart { selected_payment_method { code } } }
  placeOrder(input: { cart_id: "kDyOEKkJmIOxa6H3ceus6MjMaSF9lao1" }) {
    order { order_number }
  }
}
```

For REST, POST to `/V1/carts/mine/payment-information` (guest: `/V1/guest-carts/:cartId/payment-information`) with the same `additional_data`:

```
{
    "paymentMethod": {
        "method": "paradoxlabs_cybersource",
        "additional_data": {
            "transient_token": "eyJ...",
            "save": true
        }
    }
}
```

To pay with an already-stored card instead, send `card_id` (the card hash) and `cc_cid` (if the security code is required) in place of `transient_token`, in the same request as the place-order call.

How-To: API 3D Secure Flow

Version 4.0 runs 3D Secure 2 natively on the CyberSource REST API through three server operations — setup, authenticate, and finalize — exposed both as GraphQL mutations (cyberSourcePayerAuth*) and as REST endpoints:

```
POST /V1/carts/mine/paradoxlabs-cybersource/payer-auth/{setup|authenticate|finalize}
POST /V1/guest-carts/:cartId/paradoxlabs-cybersource/payer-auth/{setup|authenticate|finalize}
```

The authentication result is stored against the cart and instrument server-side; order placement validates it automatically, and the client never handles the CAVV or an authentication JWT. No CardinalCommerce library or credentials are involved. Note: only GraphQL can supply a cross-origin (separate-storefront) returnUrl — a REST caller gets the store-hosted or same-host return URL. Run this sequence after collecting a transient token (or selecting a stored card) and before placing the order.

Step 1: Setup and device data collection

Call `cyberSourcePayerAuthSetup` with the cart ID and exactly one of `transientToken` (new card) or `cardHash` (stored card). If it returns `skipped: true`, place the order directly. Otherwise it returns `accessToken` and `deviceDataCollectionUrl`. Collect device data by form-POSTing the access token to that URL inside a hidden (0×0) iframe, in a single hidden input named JWT. This is best-effort — allow roughly ten seconds, then continue regardless of outcome.

```
var f = document.createElement('form');
f.method = 'POST';
f.action = deviceDataCollectionUrl;   // from setup
f.target = 'ddc-iframe';              // a hidden 0x0 iframe
var i = document.createElement('input');
i.type = 'hidden'; i.name = 'JWT'; i.value = accessToken;
f.appendChild(i); document.body.appendChild(f); f.submit();
```

Step 2: Authenticate

Collect browser data on the client and call `cyberSourcePayerAuthAuthenticate` with the cart ID, that `browserInfo`, and — for a separate-origin storefront — an absolute https `returnUrl` that matches a configured Payer Auth Return URL Origin. The result status is one of:

- `success` — authenticated (or attempted); place the order.
- `skipped` — 3D Secure does not apply; place the order.
- `failed` — authentication failed; do not place the order. The card is blocked until it authenticates (see Behavior Notes).
- `challenge` — an issuer challenge is required; the result also includes `stepUpUrl` and `accessToken` (and, for reference, `acsUrl` and `pareq`). Continue to Step 3.

Step 3: Present the challenge

Run the challenge by form-POSTing the access token to `stepUpUrl` inside a visible iframe or modal, again in a single hidden input named JWT. (The built-in checkout frames a same-origin wrapper at `pdl_cybs/payerauth/challenge` and passes the handles by `postMessage` rather than in a URL; a custom integration can post directly to `stepUpUrl`.) The issuer ACS posts back to your returnUrl when the customer finishes.

```
var f = document.createElement('form');
f.method = 'POST';
f.action = stepUpUrl;                 // from authenticate
f.target = 'challenge-iframe';        // a visible iframe/modal
var i = document.createElement('input');
i.type = 'hidden'; i.name = 'JWT'; i.value = accessToken;
f.appendChild(i); document.body.appendChild(f); f.submit();
```

Step 4: Finalize

After the challenge returns, call `cyberSourcePayerAuthFinalize` with the cart ID to retrieve the final status (`success` or `failed`). The attempt is identified server-side by the cart, so no other input is needed.

Step 5: Place the order

Place the order exactly as in the checkout flow (`transient_token`, or `card_id` plus `cc_cid`). The stored Payer Authentication result is validated and consumed server-side. If Require Payer Authentication is enabled and no attempt was made, placement is refused. 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