Contents

Authorize.net Apple Pay & Google Pay **1**
1. Overview . . . 2
Key Features . . . 2
Compatibility . . . 2
2. Installation . . . 3
Via Composer . . . 3
Verify . . . 3
Uninstall . . . 3
3. Before You Start . . . 3
3.1 Authorize.net . . . 3
3.2 Google Pay . . . 4
3.3 Apple Pay . . . 4
3.4 What the Admin Shows . . . 5
4. Configuration . . . 6
4.1 Authorize.net Account . . . 6
4.2 Apple Pay . . . 8
4.3 Google Pay . . . 11
4.4 Configuration Check . . . 12
4.5 Debug Log . . . 13
5. How It Works at Checkout . . . 14
5.1 Google Pay . . . 14
5.2 Apple Pay . . . 14
5.3 When a wallet is hidden . . . 15
5.4 What happens to payment data . . . 15
6. Orders in the Admin . . . 15
6.1 Capture, refund, void . . . 15
6.2 The two Authorize.net limits . . . 16
6.3 Payment Review . . . 16
7. The Apple Pay Certificate . . . 16
8. Use Cases . . . 17
9. Troubleshooting . . . 17
9.1 A wallet button does not appear . . . 17
9.2 The payment is declined after the wallet sheet closes . . . 18
9.3 The Apple Pay sheet closes with “Apple Pay is not available right now” . . . 18
9.4 Saving the Merchant Identity is refused . . . 19
9.5 Apple cannot verify the domain . . . 20
9.6 An online refund is refused . . . 20
10. Support & Changelog . . . 20

Authorize.net Apple Pay & Google Pay

User Guide
Version: 1.0.0
Compatibility: Adobe Commerce and Magento Open Source 2.4.7 – 2.4.9, PHP 8.2 – 8.5
Vendor: Webmaster Ramos

1. Overview

Shoppers who pay with a phone or a browser wallet expect to see Apple Pay and Google Pay at checkout. A store that takes cards through Authorize.net usually runs a card module that offers neither, and the Marketplace listings that do are full card modules: adopting one means migrating stored cards, re-pointing webhooks and testing every checkout customisation again.

Authorize.net Apple Pay & Google Pay adds the two wallets to Adobe Commerce and Magento Open Source and pays them through the Authorize.net account the store already has. It installs beside your card module, reads none of its configuration and changes none of its behaviour. A wallet payment is an ordinary Authorize.net transaction, so your reports, settlement and fraud filters carry on as before.

Key Features

- **Google Pay** at the native Magento checkout, with Google's own button, offered only where Google says the shopper can pay.
- **Apple Pay** at the native Magento checkout, in Safari and, through Apple's scan-a-code flow, in desktop Chrome, Edge and Firefox.
- **Magento's checkout rules first.** Required fields and checkout agreements are checked before a wallet sheet opens; the order completes on Magento's success page.
- **The whole order lifecycle in the admin:** capture on invoice, online refunds, void, and Accept or Deny for transactions the Authorize.net Advanced Fraud Detection Suite holds.
- **A currency guard** that hides the wallets on any order in a currency the Authorize.net account does not settle.
- **Configuration Check** – a button that tests the saved settings and never reports a pass it cannot prove.
- **Apple Pay housekeeping:**
  - the domain association file served by Magento;
  - certificate expiry warnings in the admin;
  - Apple Pay withdrawn automatically while the certificate is expired.
- **Debug Log** for setup: a redacted log of every gateway call, and the reason a wallet is hidden in the browser console.
- **No payment data kept.** The wallet payload is used once and removed before anything is saved; no log can contain it.

Compatibility

Platform	Adobe Commerce and Magento Open Source 2.4.7 – 2.4.9
PHP	8.2 – 8.5
Storefronts	Luma, Hyvä Theme (with native Magento checkout)
Not supported	Hyvä Checkout (Magewire), headless and GraphQL storefronts, multishipping, third-party one-step checkouts, orders created in the admin

Google Pay has been verified end to end against the Authorize.net sandbox with a Google test card. Apple Pay has not yet been verified with a live Apple Pay payment: it is covered by automated browser tests in which Apple's script is simulated.

---

2. Installation

Via Composer

```
composer require webmaster-ramos/module-authnet-wallets
bin/magento module:enable WebRamos_AuthnetWallets
bin/magento setup:upgrade
bin/magento setup:di:compile
bin/magento cache:flush
```

Verify

```
bin/magento module:status WebRamos_AuthnetWallets
```

Then open **Stores → Configuration → Sales → Payment Methods**. The **Authorize.net Wallets** group (the extension's name in the admin) is listed with the other payment methods, with three parts: Authorize.net Account, Apple Pay and Google Pay.

Uninstall

Keep the extension installed while wallet orders still need a capture, refund or void: those operations run through it. Switch both wallets off first, then:

```
bin/magento module:disable WebRamos_AuthnetWallets
bin/magento setup:upgrade
bin/magento cache:flush
composer remove webmaster-ramos/module-authnet-wallets
```

The extension creates no database tables. Its settings stay in `core_config_data` under `payment/webramos_authnet_wallets/*`, `payment/webramos_applepay/*` and `payment/webramos_googlepay/*`.

---

3. Before You Start

Each wallet needs something from Authorize.net and something from Google or Apple. Set these up first; the Magento side takes minutes once they exist.

3.1 Authorize.net

- **API Login ID and Transaction Key** – in the Merchant Interface under Account → Settings → Security Settings → API Credentials & Keys. If your card module already uses a pair, you can use the same one.

- **A supported processor.** Authorize.net offers wallets only on processors that support payment network tokenisation. Its support articles list them (Chase Paymentech, Global Payments, TSYS, FDC Nashville, NAB EPX and Worldpay among them). Your processor is shown under Account → Merchant Profile.
- **Digital Payment Solutions.** Wallets are enabled per account under Account → Digital Payment Solutions. Only the Account Owner can sign up.
- **The Payment Gateway ID** for Google Pay. It is the Authorize.net value Google needs to encrypt a payment for your account; it is **not** the API Login ID. Where the Merchant Interface shows it depends on the interface version, so if you cannot find it, Authorize.net support will give it to you.
- **The Transaction Details API** (recommended). Under Account → Settings → Security Settings → Transaction Details API. It is off by default; online refunds use it when an order lacks the card's last four digits (section 6.2).

Test everything with an Authorize.net sandbox account first: the Environment setting switches the whole extension, Google included, between sandbox and production.

3.2 Google Pay

- **In the sandbox nothing is needed from Google.** Google's TEST environment works without registration and returns test cards.
- **For production**, Google approves your site first:
  1. Sign in to the Google Pay & Wallet Console and create a web integration for your store's domains.
  2. Submit it for review. Google checks the checkout with the Google Pay button working in the TEST environment.
  3. Once approved, the console shows your **Google Merchant ID**.
- The extension uses Google's gateway integration: Google encrypts the payment for Authorize.net directly. The KeySet ID and public key on Authorize.net's Google Pay sign-up page belong to Google's other, direct integration and are not used here.

3.3 Apple Pay

Apple Pay needs an Apple Developer account (Account Holder or Admin role) and four things from it:

1. **A Merchant ID**, for example `merchant.com.example.store` (Certificates, Identifiers & Profiles → Identifiers → Merchant IDs).
2. **The Payment Processing Certificate**, through Authorize.net: in the Merchant Interface open Account → Digital Payment Solutions, click Sign Up in the Apple Pay section, enter the same Merchant ID, click Generate Apple CSR, and upload that CSR to Apple on the Merchant ID's page. Use the CSR Authorize.net generates, not one of your own: this certificate is what lets Authorize.net decrypt Apple Pay payments.
3. **The Merchant Identity Certificate**, which the store itself uses to prove to Apple who it is. On the Merchant ID's page, under Apple Pay Merchant Identity Certificate, upload a CSR you generated yourself (RSA 2048) and download the certificate. Keep the private key that belongs to that CSR: the extension needs both (section 4.2).
4. **Your domains, registered and verified.** Add every domain that serves the checkout to the Merchant ID. Apple gives you a domain association file to verify each one. Paste it into

the extension before you click Verify: Magento serves it at the address Apple checks (section 4.2).

The Merchant ID itself never expires, but the Merchant Identity Certificate and the domain verification do. Section 7 covers the certificate.

3.4 What the Admin Shows

Stores → Configuration → Sales → Payment Methods → Authorize.net Wallets. The group opens with a header naming the extension and its version, with links to this documentation and to support; quote that line in a support request.

Every setting applies per website; the Domain Association File can also be set per store view. Save the section before you run the Configuration Check: the check reads the stored settings, not the form.

---

4. Configuration

4.1 Authorize.net Account

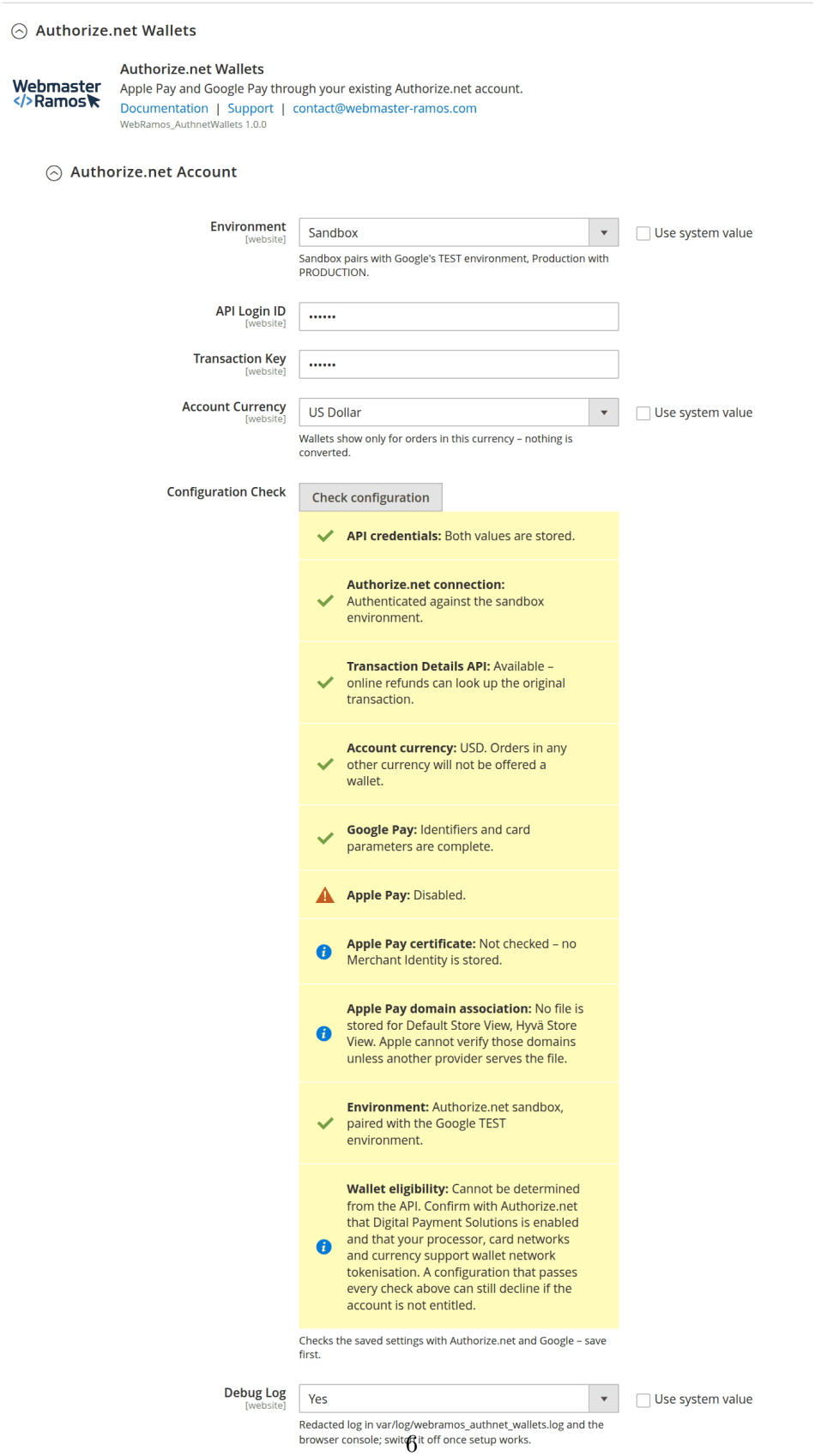


Figure 1: The Authorize.net Account group with the Configuration Check

Setting	Default	What it does
Environment	Sandbox	Sandbox or Production. It also selects Google's TEST or PRODUCTION environment, because a production Google payment cannot be decrypted by the sandbox.
API Login ID	–	Stored encrypted.
Transaction Key	–	Stored encrypted.
Account Currency	USD	The one currency this Authorize.net account settles in.
Configuration Check	–	A button; see section 4.4.
Debug Log	No	See section 4.5.

Account Currency deserves a word. An Authorize.net account settles in one currency, and Authorize.net does not reject an amount in another one: it charges the number it receives in the account's own currency. A €50 order sent as is would become a $50 charge with nothing in the answer to say so. The extension therefore offers the wallets only on orders whose currency equals Account Currency, and refuses the payment if that check is bypassed. Nothing is converted. A second currency needs a second website with its own Authorize.net account.

4.2 Apple Pay

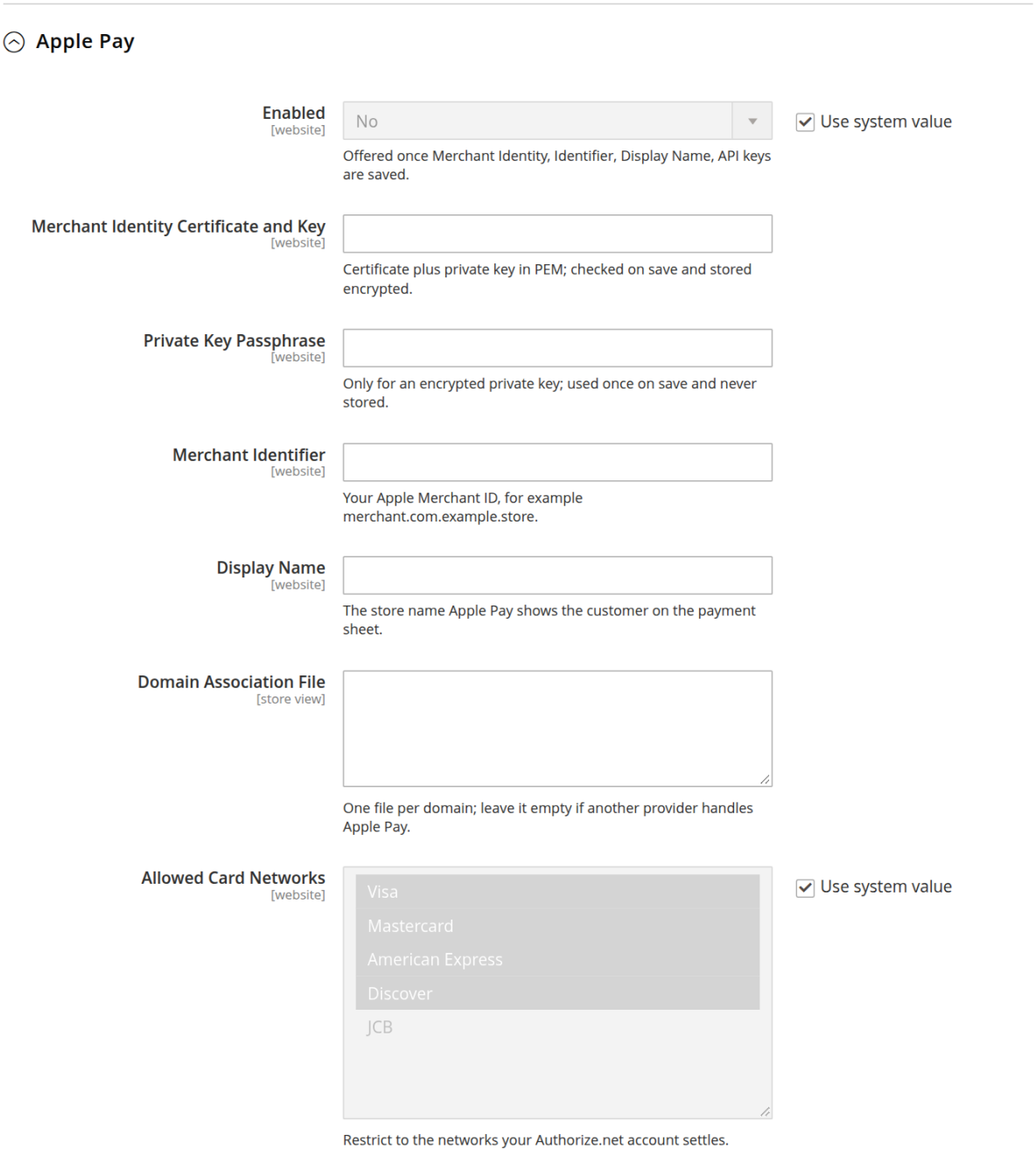


Figure 2: The Apple Pay group

Setting	Default	What it does
Enabled	No	Apple Pay is offered only once the Merchant Identity, Merchant Identifier, Display Name and the API Login ID and Transaction Key are saved.
Title	Apple Pay	The method's name in the payment list.
Payment Action	Authorize	Authorize, or Authorize and Capture.
Merchant Identity Certificate and Key	–	The Merchant Identity Certificate and its private key, pasted together.
Private Key Passphrase	–	Only for an encrypted private key.
Merchant Identifier	–	Your Apple Merchant ID, for example `merchant.com.example.store`.
Display Name	–	The store name the Apple Pay sheet shows the shopper.
Domain Association File	–	The file Apple gave you to verify the domain.
Allowed Card Networks	Visa, Mastercard, American Express, Discover	Also JCB. Only networks your Authorize.net account settles.
Sort Order	10	Position in the payment list.

Merchant Identity Certificate and Key. Paste the certificate and its private key as one block, in PEM, with their BEGIN and END lines. Apple issues the certificate as a binary `.cer` file; convert it first:

```
openssl x509 -inform der -in merchant_id.cer -out merchant_id.pem
```

If you exported the certificate and key from Keychain as a `.p12` file:

```
openssl pkcs12 -in merchant_id.p12 -clcerts -nodes
```

Add `-legacy` for an older Keychain export. The extension checks on save that the certificate and key form a pair and stores them encrypted. After a successful save it shows the certificate's UID, its expiry date and its SHA-256 fingerprint; the UID should equal your Merchant Identifier. A certificate that has already expired is refused, and one that expires within 60 days is saved with a warning.

Private Key Passphrase. Enter it only if the private key is encrypted. It is used once, to unlock the key when you save, and never stored. Leave it empty for an unencrypted key.

Domain Association File. Paste the file exactly as Apple (or your payment provider) issued it. Magento then serves it at `/.well-known/apple-developer-merchantid-domain-association` and at the same address ending in `.txt`, on this scope's domains, ahead of any other module answering

those addresses. Apple binds a file to one domain: when a website serves several domains, set the file per store view. Leave it empty where another provider handles Apple Pay; Magento then leaves both addresses to whatever answered them before. If a full-page cache may hold a “not found” answer for either address, flush it once after saving.

4.3 Google Pay

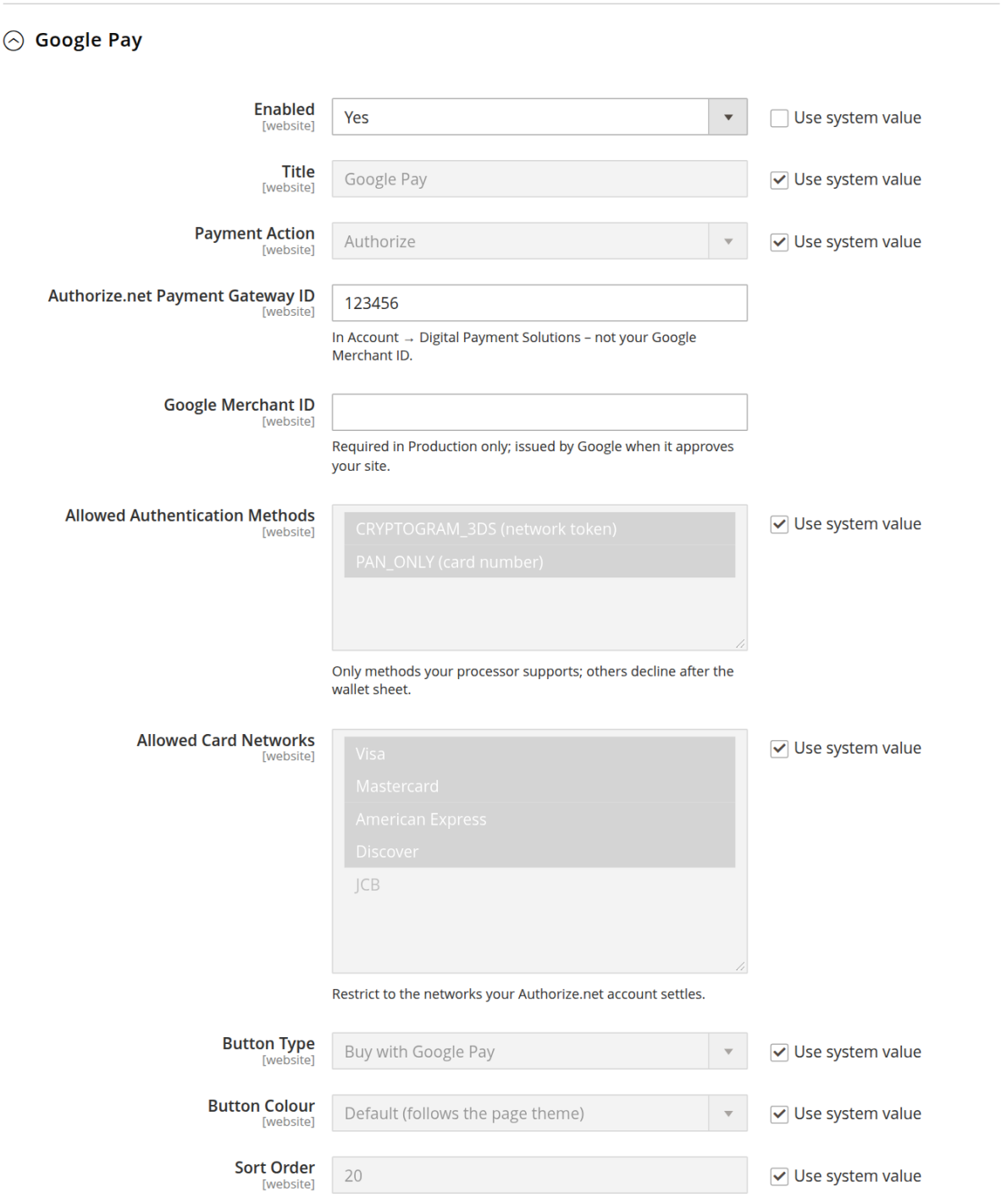


Figure 3: The Google Pay group

Setting	Default	What it does
Enabled	No	

Setting	Default	What it does
Title	Google Pay	The method's name in the payment list.
Payment Action	Authorize	Authorize, or Authorize and Capture.
Authorize.net Payment Gateway ID	–	The Authorize.net value Google encrypts the payment for (section 3.1).
Google Merchant ID	–	From the Google Pay & Wallet Console. Required in Production only.
Allowed Authentication Methods	CRYPTOGRAM_3DS, PAN_ONLY	The card types Google may offer.
Allowed Card Networks	Visa, Mastercard, American Express, Discover	Also JCB. Only networks your Authorize.net account settles.
Button Type	Buy with Google Pay	Also Pay, Checkout, Order, or Plain (logo only).
Button Colour	Default (follows the page theme)	Or Black, or White.
Sort Order	20	Position in the payment list.

The two identifiers are different values, and swapping them is the most common setup error. The Payment Gateway ID comes from Authorize.net and tells Google which processor to encrypt the payment for. The Google Merchant ID comes from Google and identifies your approved site. With a wrong Payment Gateway ID the Google Pay sheet completes normally and every payment then declines.

Allowed Authentication Methods. CRYPTOGRAM_3DS covers cards tokenised on a phone; PAN_ONLY covers cards saved to a Google account, which is what a desktop browser offers. With CRYPTOGRAM_3DS alone, Google Pay is not offered in desktop Chrome. Keep both unless your processor refuses one: a method the processor does not accept fails like a wrong Payment Gateway ID, after the sheet.

The store name on the Google Pay sheet is taken from General → Store Information → Store Name, and from the store's group name when that is empty.

4.4 Configuration Check

The button under the account settings tests the saved configuration of the website in scope. Each line reports one of four states – pass, warning, failure or unknown – and unknown is never counted as a pass.

Line	What it tells you
API credentials	Both values are stored. If not, the next two lines are skipped.
Authorize.net connection	The credentials authenticate against the selected environment.

Line	What it tells you
Transaction Details API	Whether online refunds can look up an original transaction (section 6.2). A warning tells you where to enable it.
Account currency	The currency set, and that orders in any other currency get no wallet.
Google Pay	Disabled, or what is missing: the Payment Gateway ID, a card network, or the Google Merchant ID in Production.
Apple Pay	Disabled, or every setting still missing before Apple Pay can be offered.
Apple Pay certificate	The certificate's expiry and days left; a warning from 60 days before it, a failure once it has expired or cannot be read; whether its UID matches the Merchant Identifier.
Apple Pay domain association	The store views of this website with no file stored. It never reports a pass: only your Apple Developer account can confirm that Apple verified a domain.
Environment	The Authorize.net and Google environments in use, which always move together.
Wallet eligibility	Always unknown: whether your account is entitled to wallet payments cannot be read through the API. It lists what to confirm with Authorize.net.

A configuration that passes every line can still decline if the account is not entitled to wallet payments. That is why the last line exists.

4.5 Debug Log

With **Debug Log** set to Yes, the extension writes two things:

- **A server log**, `var/log/webramos_authnet_wallets.log`: every call to Authorize.net and Apple, and every refused Apple Pay merchant validation with its reason. Wallet payloads, credentials, certificates and full upstream answers are redacted before writing; there is no raw mode.
- **Lines in the browser console** at checkout, saying why a wallet is not offered (section 9.1). The shopper sees nothing and nothing is sent anywhere; with Debug Log off the console stays silent.

Switch it off once setup works. Errors (an Apple refusal, a failed request) are logged whether Debug Log is on or not.

---

5. How It Works at Checkout

5.1 Google Pay

1. At the payment step, the extension asks Google whether this shopper can pay (`isReadyToPay()`). If not, Google Pay is not shown at all.
2. The shopper selects Google Pay and clicks Google's button. Magento first checks the required fields and the checkout agreements, then Google's sheet opens with the order's total.
3. The shopper chooses a card. Google returns an encrypted payment for Authorize.net, and Magento places the order: the authorization (or authorization and capture) runs, and the order completes on Magento's success page.

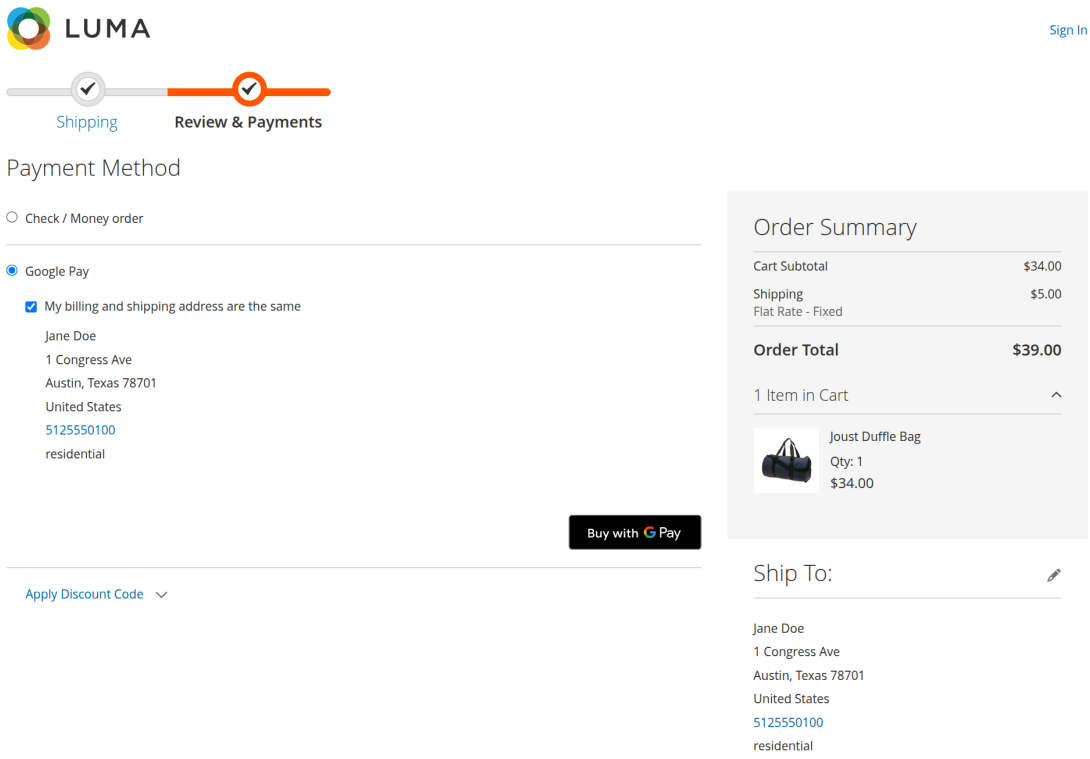


Figure 4: Google Pay at the payment step (a Hyvä Theme store with the native Magento checkout shows the same page)

5.2 Apple Pay

1. The extension loads Apple's script and asks the browser whether it can pay with Apple Pay. Safari answers directly; desktop Chrome, Edge and Firefox show Apple's button and a code the shopper scans with an iPhone.
2. When the sheet opens, Apple asks the store to prove its identity. The browser sends Apple's request to the store, which calls Apple with the Merchant Identity Certificate and hands Apple's answer back (merchant validation).
3. The shopper confirms with Face ID or Touch ID, and Magento places the order as for Google Pay.

The store answers merchant validation only on its own secure domain, only while Apple Pay is configured and the cart has items, and at most 10 times a minute per shopper session and 60 per address.

5.3 When a wallet is hidden

A wallet does not appear when:

- the order's currency differs from Account Currency;
- the wallet is disabled, or a required setting is missing (the Configuration Check names it);
- Google Pay has no Payment Gateway ID, or no card network is allowed;
- Google says the shopper cannot pay with the methods and networks allowed;
- the browser cannot use Apple Pay, or the Apple Pay certificate has expired or cannot be read;
- the page is not served over trusted HTTPS: browsers switch wallets off on an untrusted certificate.

With Debug Log on, the browser console says which of these applied.

5.4 What happens to payment data

- The wallet payload is used once. It exists only until the authorization has read it and is removed from the payment before anything is saved, whether the payment succeeds or not.
- The wallet type sent to Authorize.net is taken from the payment method, never from the browser, so a Google Pay payload cannot be submitted as Apple Pay or the reverse.
- Nothing writes the payload to a log, and the Debug Log redacts by name and also drops any long value whatever its name.
- The API credentials and the Merchant Identity are encrypted at rest and never sent to the browser; the Merchant Identity is never written to disk, even while it is used.
- What stays on the order is what Magento keeps for a card payment: the transaction ID, the last four digits, the card network and the response codes.

This describes the extension's behaviour. It is not a statement about your PCI obligations, which remain yours.

---

6. Orders in the Admin

6.1 Capture, refund, void

- **Invoice** with "Capture Online" captures the payment. With Payment Action set to Authorize, this settles the existing authorization.
- **Credit Memo** from an online invoice refunds online.
- **Cancel** or **Void** voids a transaction that has not settled yet.

The payment information names the wallet and the card – "Card Network: Visa" and "Card Number: xxxx-1111" – in the admin order, invoice and credit memo views, the customer's account, the sales e-mails and the PDFs.

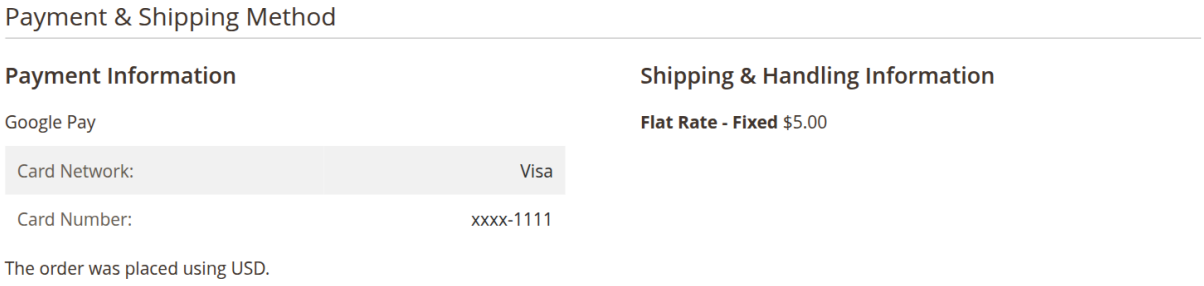


Figure 5: A Google Pay order – the payment information

6.2 The two Authorize.net limits

1. **One capture per authorization.** A partial capture consumes the whole authorization and releases the rest; a second capture is refused.
2. **Online refunds need the card's last four digits.** Authorize.net returns them at authorization and the extension stores them. If they are missing, the extension asks the Transaction Details API. If that is off too, Magento refuses the online refund and says to refund in the Merchant Interface and record an offline credit memo. It never reports a refund that did not happen.

6.3 Payment Review

When the Authorize.net Advanced Fraud Detection Suite holds a transaction, the authorization stands but needs your decision. The order goes to Payment Review and the order view offers **Accept Payment** and **Deny Payment**, which send the decision to Authorize.net. Your fraud filters keep running on wallet payments and keep receiving the billing and shipping addresses and the shopper's IP address.

A decision made directly in the Merchant Interface is not copied back to Magento; take it in the order view instead.

---

7. The Apple Pay Certificate

The Merchant Identity Certificate expires, usually about two years after Apple issues it. The extension watches the stored certificate of every website where Apple Pay is switched on:

- **From 60 days before expiry**, every admin page shows a system message naming the website, the expiry date and the days left, and saying where to paste a new certificate. A new message appears at 30 days.
- **Once it has expired**, the message says so, and Apple Pay is withdrawn: it leaves the checkout and the REST and GraphQL payment lists, the payment API refuses it, and merchant validation answers that Apple Pay is not configured. Nothing else changes; Google Pay and your other payment methods are not affected.
- **A stored certificate that cannot be read** is reported and withdrawn the same way.

To replace it: create a new Merchant Identity Certificate on the Merchant ID's page in your Apple Developer account (section 3.3), then paste it with its private key into Merchant Identity Certificate and Key and save. Apple Pay is offered again as soon as the new certificate is stored; the message goes away by itself.

Domain verification also expires. If Apple reports a domain as no longer verified, click Verify again in your Apple Developer account; the file is still served (section 4.2).

---

8. Use Cases

A store that already takes cards through Authorize.net. Install the extension, enter the same API Login ID and Transaction Key your card module uses, and enable the wallets. Stored cards, the card module's settings and your webhooks are not touched; wallet payments appear in the Merchant Interface next to card payments.

Going live safely. Configure everything with Environment set to Sandbox: Google Pay works there with Google's test cards, and the Configuration Check tests the sandbox credentials. Then switch Environment to Production, enter the production credentials and the Google Merchant ID, run the Configuration Check again and place one small real order.

One website, several domains. Apple verifies each domain separately. Set the Domain Association File per store view, register every domain under the Merchant ID, and verify each one.

Apple Pay handled by another provider on some domains. Leave the Domain Association File empty for those store views: Magento then leaves the Apple address to whatever served it before.

Capture at shipment. Set Payment Action to Authorize and invoice with "Capture Online" when the order ships. Remember the one-capture limit (section 6.2).

---

9. Troubleshooting

9.1 A wallet button does not appear

Switch **Debug Log** on, open the checkout, and open the browser's developer console. One line names the reason:

Console line	Meaning and fix
`Google Pay: hidden – no Authorize.net Payment Gateway ID is configured.`	Enter the Authorize.net Payment Gateway ID (section 4.3).
`Google Pay: hidden – isReadyToPay() answered no for this browser.`	Google says this shopper cannot pay with the allowed methods and networks. On a desktop browser, check that PAN_ONLY is allowed. The shopper may have no card in Google Pay.
`Google Pay: hidden – isReadyToPay() was rejected (…)`	Google refused the request; its status code and message follow. Usually an invalid merchant setting, or Production without Google's approval.

Console line	Meaning and fix
`Apple Pay: hidden — the Apple Pay SDK could not be loaded.`	Apple's script did not load: a network block, a content blocker, or a Content Security Policy that another extension tightened.
`Apple Pay: hidden — ApplePaySession is not available in this browser.`	The browser has no Apple Pay support.
`Apple Pay: hidden — ApplePaySession does not support version 3.`	The browser's Apple Pay is too old.
`Apple Pay: hidden — ApplePaySession.canMakePayments() answered no.`	Apple says this device cannot pay here, for example no card in Wallet.

No line at all usually means the method itself is not offered. Check that it is enabled for this website, that the order currency equals Account Currency, and run the Configuration Check. The checkout must be served over HTTPS with a certificate the browser trusts: wallets are switched off on an untrusted one.

If Google's own script fails to load, the shopper sees "Google Pay could not be loaded." as an error message at the payment step. A blocked script is usually a Content Security Policy that another extension tightened, or a content blocker.

9.2 The payment is declined after the wallet sheet closes

For security, the shopper sees only Magento's general message, "Transaction has been declined. Please try again later." The exact reason is logged:

- in `var/log/system.log`, on a line starting `Payment Error:` with Authorize.net's reason;
- with Debug Log on, in `var/log/webramos_authnet_wallets.log`, with the redacted request and answer.

The usual causes:

- a wrong **Authorize.net Payment Gateway ID** (Google encrypted the payment for another account);
- a card network or authentication method your processor does not accept;
- sandbox credentials with Environment set to Production, or the reverse;
- the account is not enabled for wallet payments (Configuration Check, Wallet eligibility).

9.3 The Apple Pay sheet closes with "Apple Pay is not available right now"

The merchant validation was refused, by the store or by Apple. With Debug Log on, the console shows `Apple Pay: merchant validation refused (HTTP <status>, <code>).`, and the server log has a line with the same code:

Code	Cause	Fix
`not_configured`	Apple Pay is not switched on or not fully configured for this store view, or its certificate has expired.	Run the Configuration Check and complete what it names.

Code	Cause	Fix
`host_mismatch`	The checkout was opened on a domain other than the store view's secure base URL.	Open the checkout on the store's own domain, or correct the secure base URL.
`url_refused`	The validation address the browser sent is not one of Apple's, or is Apple's sandbox address while Environment is Production.	Use the Apple sandbox only with Environment set to Sandbox. An Apple address refused this way is always logged as an error; any other address only with Debug Log on, since it does not come from Apple's sheet.
`no_quote`	The shopper's cart is empty or expired.	Reload the checkout.
`rate_limited`	More than 10 attempts a minute from one session, or 60 from one address.	Wait a minute. The shopper sees "Too many Apple Pay attempts. Please wait a minute and try again."
`invalid_form_key`	The request carried no valid form key: the shopper's session expired, or a cache served the checkout with a stale key.	Reload the checkout. If it persists, check that the checkout page is not full-page cached.
`validation_failed`	Apple refused the store's request.	See below.

`validation_failed` means Apple answered with an error; the server log always has the line, with Apple's status code and message. The usual causes:

- the Merchant Identity Certificate does not belong to the Merchant Identifier (the Configuration Check compares the certificate's UID);
- the domain is not registered or no longer verified under that Merchant ID;
- the certificate was revoked in the Apple Developer account.

If Apple's sheet itself refuses the session the store returned, the sheet closes with the same message.

9.4 Saving the Merchant Identity is refused

Message begins	Fix
"The certificate could not be read."	Convert Apple's `.cer` file to PEM first (section 4.2).
"Paste exactly one certificate together with its private key."	Paste one certificate and one private key, not a chain or two keys.
"The private key does not belong to this certificate."	The key is from another CSR. Use the key generated with the CSR you uploaded to Apple.
"This private key is encrypted."	Enter its passphrase in Private Key Passphrase.
"The passphrase does not unlock this private key."	Check the passphrase.

Message begins	Fix
“This private key uses the legacy encrypted PEM format”	Convert it with `openssl pkcs8 -topk8 -in key.pem -out key-pkcs8.pem`, as the message says.
“The certificate expired on”	Create a new certificate (section 7).

9.5 Apple cannot verify the domain

- Open `https://<your-domain>/.well-known/apple-developer-merchantid-domain-association` in a browser. It must show the file’s text.
- If it shows a “not found” page, check that the file is saved for the store view of that domain, and flush the full-page cache once.
- If it shows another file, another module or a web-server rule answers the address first. The extension serves it only through Magento.
- Saving a file is refused if it is an HTML or error page, a certificate, or over 65,535 bytes: paste the file exactly as issued.

9.6 An online refund is refused

The original transaction’s last four digits are missing and the Transaction Details API is off. Refund in the Merchant Interface and record an offline credit memo, or enable the Transaction Details API (section 3.1) so the next refund can look them up.

---

10. Support & Changelog

- **Author:** Webmaster Ramos
- **Email:** contact@webmaster-ramos.com – please include the line at the top of the Authorize.net Wallets group (`WebRamos_AuthnetWallets` and its version)
- **Website:** https://webmaster-ramos.com
- **Changelog:** see `CHANGELOG.md` inside the package