back to DOCUMENTATION

API Version 4

Overview
- Accessing the API
- Environments
- Payload Encryption
- SDK
- Error Handling
- Pagination
- Sorting
- HATEOAS
- What's New in V4

References

RESOURCES

Users
Business Stakeholders
Bank Accounts
Bank Cards
PayPal Accounts
Venmo Accounts
Prepaid Cards
Paper Checks
Transfer Methods
Payments
Transfers
Spendback
Spendback Refunds
Balances
Receipts
Programs
Accounts
Transfer Method Configurations
Webhook Notifications
Status Transitions
Authentication Token
Supplemental Data

Payload Encryption

Use the Payload Encryption API as a second factor of authentication.

HOW HYPERWALLET PROTECTS SENSITIVE DATA

The Payload Encryption endpoint is an alternative to IP allowlisting that you can use as a second factor of authentication. Payload Encryption uses Javascript Object Signing and Encryption (JOSE) and JSON Web Tokens (JWT) to guarantee the confidentiality, accuracy, integrity, and origin of your payment data.

STANDARDS

Hyperwallet uses the following standards in our Payload Encryption process:

- We use the Javascript Object Signing and Encryption (JOSE) framework.
- We follow the JSON Web Tokens (JWT) standard.
- We sign using JSON Web Signature (JWS) and encrypt using JSON Web Encryption (JWE)
- We use JSON Web Keys (JWK) to publish public keys for the JWTs.
- We use the RS256 algorithm by default. See JWS Algorithms for the full list of algorithms we support.

REST API AUTHENTICATION

Hyperwallet offers 2 second factor options for authentication:

- IP address allowlisting
- Layer 7 encryption using Javascript Object Signing and Encryption (JOSE) and JSON Web Tokens (JWT)

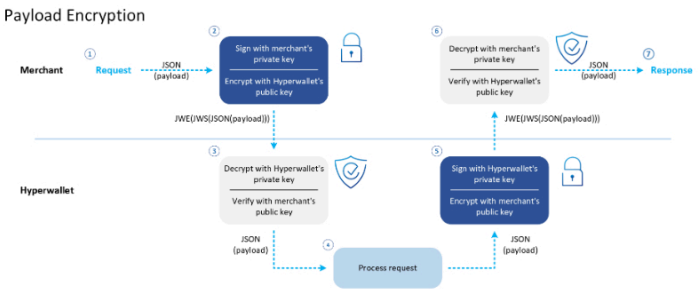
ALLOWLISTING

You need to provide source IP addresses if you choose allowlisting as the second factor for authentication. Hyperwallet will only accept API calls that originate from these IP addresses. We also support Classless Inter-Domain Routing (CIDR) notation.

PAYLOAD ENCRYPTION

Hyperwallet accepts requests from any IP address if you choose Payload Encryption as the second factor for authentication.

HOW ENCRYPTION WORKS



1. JSON is first signed using the client's private signing key.
2. It is then encrypted using one of Hyperwallet's public encryption keys:
   - Production: https://api.paylution.com/jwkset
   - User Acceptance Testing (UAT): https://uat-api.paylution.com/jwkset.
3. The request is sent to Hyperwallet with the content-type: `application/jose+json`.
4. The payload is decrypted using the corresponding Hyperwallet private encryption key.
5. The signed payload is then verified using the client's public signing key.
6. System performs the request and then generates a response that will be signed and encrypted.
7. The system looks for a match using the same algorithm used to sign in with the client's private signing key in step 1. For example:
   - The client signed the payload using a RS256 signing key.
   - The system looks in Hyperwallet's private keyset for an RS256 signing key to sign the response.
8. The system looks for a match in the hosted client public keyset using the same algorithm as the one used to encrypt Hyperwallet's public encryption key in step 2. For example:
   - The client encrypted the payload using an RSA_OAEP_256 encryption key.
   - System looks in client's hosted public keyset for an RSA_OAEP_256 encryption key to encrypt the response.

The Payload Encryption endpoint uses "Basic" HTTP authentication (BA) for all clients. You need to provide usernames and passwords for BA during onboarding.

Request payloads need to be signed and encrypted, in addition to BA and Transport Layer Security (TLS). You also need to decrypt and then validate the signatures of any responses that access the payload.

Host your public keys as a JSON Web Keys (JWK) set and provide the public JWK URL during onboarding. Use different keys for User Acceptance Testing (UAT) and

production. Host these keys at different URLs, using TLS with a valid certificate.

Hyperwallet caches your public keys when possible. However, you need to ensure that your public keys are always accessible to avoid service interruption. Hyperwallet supports public/private keys and compact JWT serialization. Hyperwallet doesn't support shared secrets or password-based encryption.

Payload Encryption signs requests and responses using JSON Web Signatures (JWS), and encrypts them using JSON Web Encryption (JWE). A payload needs to be signed then encrypted. Payloads are formatted as `JWE(JWS(payload))`. JOSE requests and responses are identical to their unencrypted representations. Use the `ContentType` header to pass the payload format.

An **encrypted request** needs to pass `application/jose+json` as the value for the `Content-Type` and `Accept` headers:

```
Content-Type: application/jose+json
Accept: application/jose+json
```

A request that isn't encrypted needs to pass `application/json` as the value for the `Content-Type` and `Accept` headers.

If the encrypted request can't be decrypted, is incorrect, or uses an unsupported algorithm, the Payload Encryption endpoint rejects the request with a response that includes `Content-Type: application/jose+json`.

When the encrypted request uses an unsupported algorithm or an incorrect token, the Payload Encryption endpoint rejects the request with a `4xx HTTP` response.

Hyperwallet updates the list of supported algorithms based on the latest security best practices.

HOW DECRYPTION WORKS

This algorithm sample shows how to construct a request using the HS256 algorithm and a JSON Web Token (JWT).

ALGORITHM EXAMPLE

Header: algorithm token type

```
{
  "alg": "HS256",
  typ": "JWT",
  "kid": "01-2822"
}
```

Payload: data

```
{
  "token" : "usr-f9154e16-94e8-4686-a84e-a75688ace7b5",
  "status" : "PRE_ACTIVATED",
  "verificationstatus" : "NOT_REQUIRED",
  "createdOn" : "201910-30T22:15:45",
  "c1ientUserId" : "CSK7b8Ffch",
  "profi1eType" : "INDIVIDUAL",
  "firstName" : John",
  "lastName" : "Smith",
  dateOfBirth" : "1980-01-01",
  "email" : "john@company.com",
  "addressLine1" : "123 Maln Street",
  "city" : "New York",
  "stateProvince" : "NY",
  "country" : "US",
  "postalCode" : "10016",
  "language" : "en",
  "programToken": "prg-83836cdf-2ce2-4696-8bc5-f1b860777238c",
  "links" : [
    {
      "params" {
        "rel" : :self"
      },
      "href" : "https://uat-api.paylution.com/rest/v4/users/usr-
      f9154e16-94e8-4686-a84e-a75688ace7b5"
      }
    ]
}
```

Verify signature

```
HMACSHA256(
  base64UrlEncode(header) +
  base64UrlEncode(payload) ,
  your-256-b1t-secret
) secret base64 encoded
```

SIGNATURE

Use your private key to sign request payloads.

Hyperwallet uses your public JSON Web Key (JWK) set to validate the signature on payout requests. We look for this JWK at the URL that you provide during onboarding. The Basic HTTP authentication process passes the JWK set URL with the request user's details.

The Payload Encryption endpoint ignores any `jku` and `jwk` headers used with JWK public keys.

Hyperwallet signs response payloads with our private key.

The Payload Encryption response passes a `kid` header to identify the public key that you use to validate the signature. The `jti` header serves as a nonce with JWT and passes a UUID.

> **Note:** It isn't practical to validate a nonce for all solutions, so you and Hyperwallet can choose whether or not to validate this nonce.

JWS ALGORITHMS

Payload Encryption supports the following JWS algorithms:

- RSASSA-PKCS1-v1_5: RS256, RS384, RS512
- RSASSA-PSS: PS256, PS384, PS512
- ECDSA: ES256, ES384, ES512

EXPIRY

EXPIRY

Payload Encryption uses the `exp` header with JWS to prevent token reuse. The `exp` header is defined as a part of the JSON Web Token (JWT), not part of the JWS. Use the `crit` header to show that Payload Encryption needs to process the `exp` header.

Signatures expire 5 minutes after generation.

JSON WEB ENCRYPTION

The content encryption keys (CEK) in a request need to be encrypted with Hyperwallet's public key.

Hyperwallet's public keys are available at https://api.paylution.com/jwkset (Production) and https://uat-api.paylution.com/jwkset (UAT). JWE headers need to pass a key identifier (`kid`) which can be used to locate the appropriate key from the JWK public key set.

> **Note:** UAT and Production environments use different keys.

Hyperwallet uses your public key to encrypt the Content Encryption Key (CEK) in a response. Decrypt the CEK using your private key, and then use that CEK to decrypt the response payload.

Hyperwallet uses the same algorithm as the request if an appropriate public key can be found at the JWK at the URL you provided.

JWE ALGORITHMS

Payload Encryption supports the following JWE algorithms. The `alg` parameter in the JWE header passes the algorithm:

- RSA-OAEP-256
- ECDH-ES: ECDH-ES+A128KW, ECDH-ES+A192KW, ECDH-ES+A256KW

JWE ENCRYPTION ALGORITHMS

Payload Encryption supports the following JWE Encryption algorithms. The `enc` parameter in the CEK header passes the encryption algorithm:

- AES_CBC_HMAC_SHA2: A128CBC-HS256, A192CBC-HS384, A256CBC-HS512
- AES GCM: A128GCM, A192GCM, A256GCM

PUBLISHING PUBLIC KEYS

Publish your public keys at the URL that you provided for the JWK when onboarding. You may publish these keys as a JWK keyset. Use DNS-compliant bucket names when publishing to an Amazon S3 bucket.

KEY ROTATION

You can rotate your keys without coordinating with Hyperwallet because you control the publication of your public keys at your own URL:

1. Update your keyset to include your new and old keys.
2. Update your application to use the new key ID in all active requests.
3. Remove the old key from the keyset and republish with only the new key.

Publish your keys to the same URL that you provided to Hyperwallet during onboarding. You can change the JWK URL by coordinating with Hyperwallet's production support team.

When we refresh keys, we publish the key at least 1 hour before using the key. If a merchant fetches and caches a JWK at least once every hour, they are unlikely to use an uncached key.

Hyperwallet fetches and caches a merchant's JWK keys every hour. When a merchant refreshes their keys, they need to wait at least 1 hour before using that key in the payload.

Hyperwallet rotates their keys once per year, and encourages you and your merchants to do the same.

ERROR MESSAGES

Possible error messages a client might receive when integrating. These are wrapped in our standard error message JSON.

```
{
  "errors" : [ {
    "message" : "JWS signature is expired. crit-exp header was in the past.",
    "code" : "JWT_ERROR"
  } ]
}
```

Status code	Error message	Explanation
`400`	User is configured to use only JWT JOSE encrypted messages. Expected contentType or accept header value is `application/jose+json`	The client tried to send a plaintext json request. They must encrypt everything if they are configured for JOSE.
`400`	Signature could not be verified	The signature the client used was not valid.
`400`	Payload not a signed JWS Object	The client encrypted something other than a signed JWS object.
`400`	Only JWE Objects are permitted	The client sent a JOSE object that is not a JWE object.
`500`	No Hyperwallet JWK candidate was found to sign the response so the request not fulfilled	The client attempted to use a signing algorithm which we do not support. They should refer to the reference doc for our supported list.
`500`	No JWK found in the client key set which matches the requested encryption method and algorithm so the request was not fulfilled.	The client requested an algorithm which they themselves do not support.
`400`	JWE Encryption algorithm (`enc` header) " + `encryptionMethod` + "is not supported Algorithm (`alg` header) " + `jweAlgorithm` + "is not supported for JWE	The client attempted to use an algorithm which we do not support. They should refer to the reference doc for our supported list.