
Workgroup: Network Working Group
Internet-Draft: draft-card-charge-00
Published: 18 May 2026
Intended Status: Informational
Expires: 19 November 2026
Author: J. Brans
Visa

Card Network Charge Intent for HTTP Payment Authentication

Abstract

This document defines the "card" payment method and its implementation of the "charge" intent within the Payment HTTP Authentication Scheme. It specifies how clients and servers exchange one-time card payments using encrypted network tokens.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 19 November 2026.

Copyright Notice

Copyright (c) 2026 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

This document may not be modified, and derivative works of it may not be created, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

1. Introduction	3
1.1. Card Charge Flow	3
1.2. Relationship to the Payment Scheme	4
2. Requirements Language	4
3. Terminology	5
4. Intent Identifier	5
5. Intent: "charge"	5
6. Request Schema	6
6.1. Shared Fields	6
6.2. Method Details	7
6.3. Encryption Key	8
7. Credential Schema	9
7.1. Credential Structure	9
7.2. Payload Fields	10
7.3. Encrypted Payload Format	11
7.4. Token Data	12
7.5. Dynamic Data	13
7.6. Billing Address Data	14
8. Verification Procedure	14
8.1. Challenge Binding	15
8.2. Idempotency and Replay Protection	15
9. Settlement Procedure	16
9.1. Receipt Generation	16
10. Security Considerations	17
10.1. Transport Security	17
10.2. Credential Security	17

10.3. Billing Data Handling	17
11. IANA Considerations	18
11.1. Payment Method Registration	18
11.2. Payment Intent Registration	18
12. References	18
12.1. Normative References	18
12.2. Informative References	19
Appendix A. Client Enabler Profile	19
A.1. Token Request Interface	20
Appendix B. ABNF Collected	21
Appendix C. Examples	21
C.1. Charge Example (HTTP Transport)	21
Appendix D. Acknowledgements	24
Author's Address	24

1. Introduction

This specification defines the "card" payment method for use with the "charge" intent [[I-D.payment-intent-charge](#)] in the Payment HTTP Authentication Scheme [[I-D.httppauth-payment](#)]. The charge intent enables one-time card payments where the server processes the payment immediately upon receiving the credential.

The card method is PSP-agnostic. Client Enablers **SHOULD** provision agent-specific payment tokens using network services such as [[VISA-INTELLIGENT-COMMERCE](#)].

1.1. Card Charge Flow

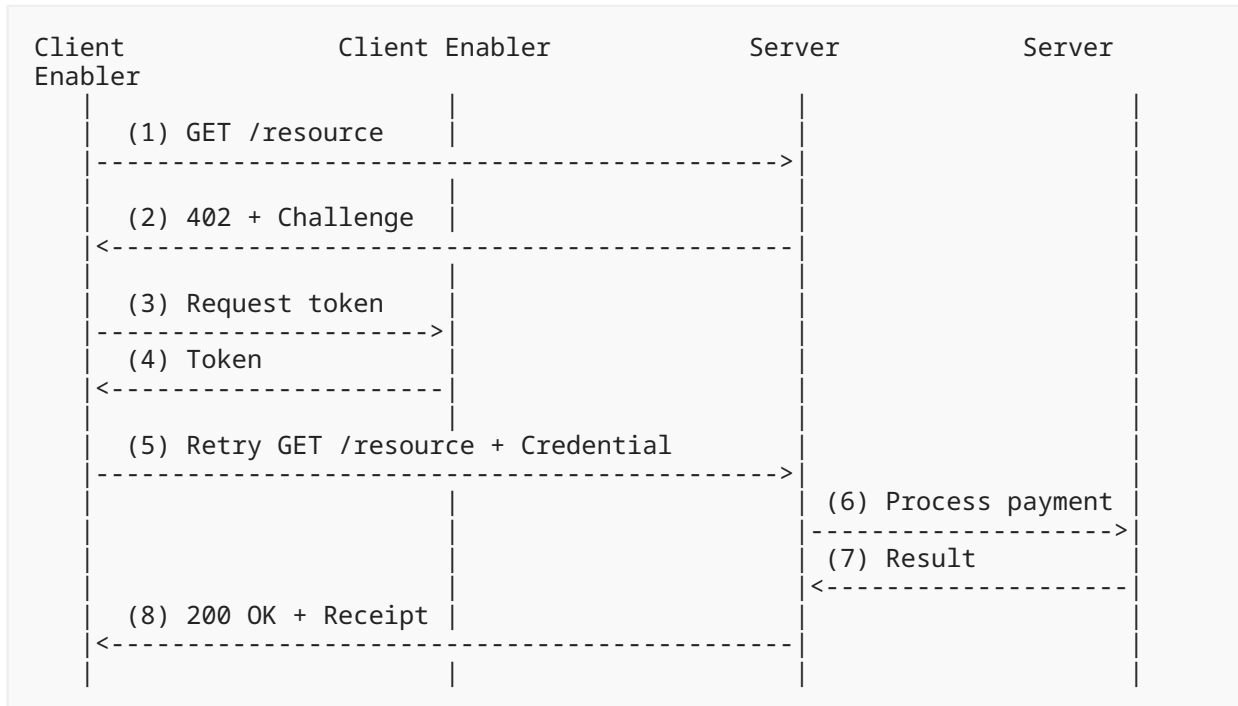
The card method implements the charge intent flow defined in [[I-D.payment-intent-charge](#)]:

1. Client requests a resource.
2. Server responds 402 with a Challenge (amount, currency, accepted networks, encryption key).
3. Client forwards challenge context to its Client Enabler.
4. Client Enabler provisions a network token, encrypts it with the server's key, and returns the credential.

5. Client retries the request with an Authorization: Payment header containing the encrypted credential.
6. Server forwards the credential to its Server Enabler.
7. Server Enabler decrypts and processes the payment.
8. Server returns 200 with Payment-Receipt and the resource.

The client may include Trusted Agent Protocol [VISA-TAP] signature headers for additional identity assurance.

The following diagram illustrates the flow:



1.2. Relationship to the Payment Scheme

This document is a payment method specification as defined in [I-D.httpauth-payment]. It implements the "charge" intent defined in [I-D.payment-intent-charge] for the "card" payment method. It defines the methodDetails, credential payload, verification, and settlement procedures specific to card payments.

2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Terminology

Payment Data An encrypted token and associated metadata produced by a Client Enabler.

Client Enabler (CE) See [Appendix A](#).

Payment Service Provider (PSP) An entity that provides payment processing services, including transaction authorization, settlement, and related functions.

Vault Provider Entity that stores sensitive card material and mediates calls to token service providers. A vault provider is one type of Client Enabler.

Token Service Provider (TSP) Entity that provisions network tokens and cryptograms. Could be a PSP, card network, issuer processor, or a vault provider with direct network connections.

Server Enabler A PSP or processing entity on the server side that decrypts and processes the encrypted network token credential.

Network Token A card-network-issued token used for transaction processing. Never exposed to the client or server.

Cryptogram A one-time-use value generated alongside a network token to authenticate a specific transaction.

4. Intent Identifier

This specification defines the following intent for the card payment method:

```
charge
```

The intent identifier is case-sensitive and **MUST** be lowercase.

5. Intent: "charge"

This specification implements the "charge" intent defined in [\[I-D.payment-intent-charge\]](#). A one-time card payment of the specified amount. The server processes the payment immediately upon receiving the credential.

The charge intent properties (payment timing, idempotency, reversibility) are defined in [\[I-D.payment-intent-charge\]](#). For the card method, reversibility is subject to card network chargeback rules.

Fulfillment mechanism: The client obtains an encrypted network token from its Client Enabler and submits it in an Authorization: Payment header. The Server Enabler decrypts and processes the payment through existing card network rails.

6. Request Schema

The server issues an HTTP 402 response with a WWW-Authenticate: Payment header per [I-D.httpauth-payment]. The request parameter is a base64url-encoded JSON object containing the shared fields defined in [I-D.payment-intent-charge] and card-specific extensions in the methodDetails field.

Example:

```
HTTP/1.1 402 Payment Required
WWW-Authenticate: Payment
  id="ch_9xK2mR4vB7nQ",
  realm="api.merchant.com",
  method="card",
  intent="charge",
  expires="2026-02-19T12:10:00Z",
  request="eyJhbW91bnQiOiI0OTk5IiwiaY3VyY3kiOiJ1c2Qi..."
Cache-Control: no-store
```

6.1. Shared Fields

Field	Type	Required	Description
amount	string	REQUIRED	Amount in smallest currency unit (e.g., "4999" = \$49.99).
currency	string	REQUIRED	ISO 4217 code, lowercase (e.g., "usd").
recipient	string	OPTIONAL	Merchant identifier as registered with the Server Enabler (acquirer). For card payments, this is the merchant ID assigned by the acquiring PSP (e.g., "merch_abc123").
description	string	OPTIONAL	Human-readable description of the payment.
externalId	string	OPTIONAL	Merchant's external reference (order ID, invoice number, etc.).

Table 1

6.2. Method Details

Field	Type	Required	Description
<code>methodDetails.acceptedNetworks</code>	array	REQUIRED	Card networks accepted (e.g., ["visa", "mastercard"]).
<code>methodDetails.merchantName</code>	string	REQUIRED	Human-readable merchant name for display (e.g., "Acme Corp").
<code>methodDetails.encryptionJwk</code>	object	CONDIT.	Embedded JWK ([RFC7517] Section 4) containing the server's RSA public encryption key. REQUIRED if <code>jwksUri</code> is absent; MUST NOT be present if <code>jwksUri</code> is present.
<code>methodDetails.jwksUri</code>	string	OPTIONAL	HTTPS URI of a JWK Set ([RFC7517] Section 5). MUST be on the same origin as the realm. When present, <code>kid</code> MUST also be present.
<code>methodDetails.kid</code>	string	CONDIT.	Key ID referencing a key in the JWKS. REQUIRED when <code>jwksUri</code> is present.
<code>methodDetails.billingRequired</code>	boolean	OPTIONAL	When true, the Client Enabler SHOULD include billing info in the credential payload. See Section 7.6 .

Table 2

Challenge expiry is conveyed by the `expires` auth-param in WWW-Authenticate per [I-D.httpauth-payment] and [I-D.payment-intent-charge]. Request objects **MUST NOT** duplicate the expiry value.

6.3. Encryption Key

The server provides its RSA public encryption key using one of two mechanisms:

1. Embedded JWK (`encryptionJwk`) -- The server includes a JSON Web Key [RFC7517] directly in `methodDetails`. This is the **RECOMMENDED** approach for most deployments. It requires no additional infrastructure and works in environments where the Client Enabler cannot make outbound HTTP calls.
2. JWKS URI (`jwtksUri` + `kid`) -- The server hosts a JWK Set at an HTTPS endpoint and references the key by its `kid`. The `jwtksUri` **MUST** be on the same origin as the challenge realm. This approach supports centralized key rotation and is suitable for large platforms managing keys across many merchants.

Key requirements:

- The key **MUST** be RSA (`"kty": "RSA"`) with a minimum length of 2048 bits. Servers **SHOULD** use 2048-bit or 4096-bit keys.
- The JWK **MUST** include `"alg": "RSA-OAEP-256"` and `"use": "enc"`. Client Enablers **MUST** reject keys where `alg` is not `"RSA-OAEP-256"` or `use` is not `"enc"`.
- The JWK **MUST** include a `"kid"` value.
- Server Enablers that manage encryption keys via X.509 certificates **MAY** include the `"x5c"` parameter ([RFC7517] Section 4.7) in the JWK.

Key resolution procedure:

1. If `jwtksUri` is present in `methodDetails`, the Client Enabler **MUST** verify the URI is on the same origin as the challenge realm. If the origins differ, the CE **MUST** reject the challenge. The CE **MUST** fetch the JWK Set from the URI over HTTPS and select the key matching `kid`. If the `kid` is not found, the CE **MUST** reject the challenge.
2. Otherwise, if `encryptionJwk` is present in `methodDetails`, the CE **MUST** use the embedded key directly.
3. If neither is present, the CE **MUST** reject the challenge.
4. The CE **MUST** validate the resolved key: `kty` **MUST** be `"RSA"`, `alg` **MUST** be `"RSA-OAEP-256"`, and `use` **MUST** be `"enc"`.

When `encryptionJwk` is used, the `kid` value is taken from within the JWK object. A top-level `kid` field **MUST NOT** be present when `encryptionJwk` is used. When `jwtksUri` is used, the top-level `kid` field identifies which key to select from the JWK Set.

The Server Enabler (or its delegated infrastructure) is responsible for key pair generation and private key management.

The Client Enabler **MUST** encrypt the token payload as a JWE [RFC7516] compact serialization using the key management algorithm from the JWK (`alg`, which **MUST** be `"RSA-OAEP-256"`) and content encryption algorithm AES-256-GCM (`enc: "A256GCM"`) per [RFC7518]. The JWE protected

header **MUST** include alg, enc, and kid. Implementations **MUST NOT** use PKCS#1 v1.5 padding or direct RSA encryption. The client forwards the JWE token to the server without inspecting it. The Server Enabler holds the corresponding private key and decrypts the JWE to recover the token payload for processing.

Example (embedded JWK in methodDetails):

```
{
  "amount": "4999",
  "currency": "usd",
  "recipient": "merch_abc123",
  "description": "Pro plan – monthly subscription",
  "methodDetails": {
    "acceptedNetworks": ["visa", "mastercard", "amex"],
    "merchantName": "Acme Corp",
    "encryptionJwk": {
      "kty": "RSA",
      "kid": "enc-2026-01",
      "use": "enc",
      "alg": "RSA-OAEP-256",
      "n": "0vx7agoebGcQSuu...",
      "e": "AQAB"
    }
  }
}
```

7. Credential Schema

The client retries the original request with an Authorization: Payment header containing a base64url-encoded JSON credential, following the standard MPP credential structure [[I-D.httpauth-payment](#)].

Example request:

```
GET /api/resource HTTP/1.1
Host: api.merchant.com
Authorization: Payment eyJjaGFsbGVuZ2UiOnsiaWQiOiJjaF85eEsy...
```

7.1. Credential Structure

The decoded credential follows the standard MPP format:

```

{
  "challenge": {
    "id": "ch_9xK2mR4vB7nQ",
    "realm": "api.merchant.com",
    "method": "card",
    "intent": "charge",
    "request": "eyJhbW91bnQiOiI00Tk5Ii...",
    "expires": "2026-02-19T12:10:00Z"
  },
  "payload": {
    "encryptedPayload": "<base64-encoded RSA-OAEP ciphertext>",
    "network": "visa",
    "panLastFour": "4242",
    "panExpirationMonth": "06",
    "panExpirationYear": "2028",
    "billingAddress": {
      "zip": "94102",
      "countryCode": "US"
    },
    "cardholderFullName": "Jane Smith",
    "paymentAccountReference": "PAR9876543210987654321012345"
  }
}

```

The challenge field echoes the challenge from the 402 response. The payload field contains the card-specific payment proof.

7.2. Payload Fields

The payload field largely conforms to [\[EMV-SRC-API\]](#).

Field	Type	Required	Description
encryptedPayload	string	REQUIRED	Encrypted Payload (Section 7.3). Client and server MUST NOT parse.
network	string	REQUIRED	Card network: "visa", "mastercard", "amex", "discover".
panLastFour	string	REQUIRED	Last four digits of the card number as displayed to the cardholder.
panExpirationMonth	string	REQUIRED	PAN expiration month (e.g., "06").
panExpirationYear	string	REQUIRED	PAN expiration year. MUST be four digits (e.g., "2028") when present.
billingAddress	object	CONDITIONAL	If <code>billingRequired</code> is true, Billing Address information (Section 7.6) is present.

Field	Type	Required	Description
cardholderFullName	string	OPTIONAL	Cardholder full name. Client Enablers SHOULD include when available from the TSP or the CE's cardholder records.
paymentAccountReference	string	CONDITIONAL	Also known as PAR. REQUIRED when available from the TSP. Client Enablers MUST include when the TSP provides it.

Table 3

The `encryptedPayload` field is required for payment processing, typically used by Server Enabler. The `network`, `panLastFour`, `panExpirationMonth`, and `panExpirationYear` fields may be used by Server for cardholder UX purposes. The `billingAddress` field may be used by Server for their business process (e.g., tax calculation) and may also be forwarded to Server Enabler for address verification (Section 7.6). The `cardholderFullName` field may be used for both payment as well as cardholder communication purposes. The `paymentAccountReference` may be used to identify cardholder across several channels and/or developers without storing sensitive information.

7.3. Encrypted Payload Format

The `encryptedPayload` field is a JWE [RFC7516] compact serialization containing an encrypted JSON object with all fields required to process the charge. The plaintext **MUST** be a minified UTF-8 encoded JSON object.

The JWE protected header **MUST** contain:

```
{"alg": "RSA-OAEP-256", "enc": "A256GCM", "kid": "enc-2026-01"}
```

The `alg` value **MUST** match the `alg` in the server's JWK. The `kid` value **MUST** match the `kid` of the encryption key resolved per Section 6.3. The `enc` value **MUST** be "A256GCM". The protected header **MUST NOT** include `zip` (compression); compressing before encryption can enable information leakage attacks.

The decrypted plaintext is a JSON object with two **REQUIRED** fields:

```

{
  "token": {
    "paymentToken": "4242424242424242",
    "tokenExpirationMonth": "06",
    "tokenExpirationYear": "2034",
    "eci": "07"
  },
  "dynamicData": {
    "dynamicDataValue": "AmDDBjkH/4A=",
    "dynamicDataType": "CARD_APPLICATION_CRYPTOGAM_SHORT_FORM",
    "dynamicDataExpiration": 1746296464
  }
}

```

Field	Required	Description
token	REQUIRED	Token object (Section 7.4). Client and Server MUST NOT parse.
dynamicData	REQUIRED	Dynamic data format (Section 7.5).

Table 4

The CE constructs the JWE as follows:

1. Generate a random 256-bit Content Encryption Key (CEK).
2. Encrypt the CEK with the server's RSA public key using RSA-OAEP-256 ([[RFC7518](#)] Section 4.3).
3. Encrypt the minified JSON plaintext with AES-256-GCM using the CEK and a random 96-bit IV.
4. Assemble the JWE compact serialization:
 BASE64URL(header) . BASE64URL(encryptedKey) . BASE64URL(iv) . BASE64URL(ciphertext) . BASE64URL(tag)

The Server Enabler decrypts the JWE using the corresponding RSA private key to recover the CEK, then decrypts the ciphertext with AES-256-GCM to obtain the token payload.

Clients and servers **MUST NOT** parse the `encryptedPayload` field.

7.4. Token Data

The `token` field in the `encryptedPayload` object is a JSON object with the following **REQUIRED** fields:

```
{
  "paymentToken": "4242424242424242",
  "tokenExpirationMonth": "06",
  "tokenExpirationYear": "2034",
  "eci": "07"
}
```

Field	Required	Description
paymentToken	REQUIRED	Network token number as issued by the TSP.
tokenExpirationMonth	REQUIRED	Two-digit expiration month (e.g., "06").
tokenExpirationYear	REQUIRED	Four-digit expiration year (e.g., "2028").
eci	REQUIRED	Electronic Commerce Indicator.

Table 5

7.5. Dynamic Data

The `dynamicData` field in the `encryptedPayload` object is a JSON object with the following **REQUIRED** fields:

```
{
  "dynamicDataValue": "AmDDBjkH/4A=",
  "dynamicDataType": "CARD_APPLICATION_CRYPTOGAM_SHORT_FORM",
  "dynamicDataExpiration": 1746296464
}
```

Field	Required	Description
dynamicDataValue	REQUIRED	Cryptogram issued by the TSP. MUST be provided when <code>dynamicDataType</code> is not "NONE".
dynamicDataType	REQUIRED	Cryptogram type per [EMV-SRC-API]. Valid values: "CARD_APPLICATION_CRYPTOGAM_SHORT_FORM", "CARD_APPLICATION_CRYPTOGAM_LONG_FORM", "CARDHOLDER_AUTHENTICATION_CRYPTOGAM", "NONE".
dynamicDataExpiration	REQUIRED	Unix Epoch Format.

Table 6

7.6. Billing Address Data

When `billingRequired` is true in the challenge `methodDetails`, the Client Enabler **SHOULD** include billing information in the credential payload.

The `billingAddress` field in the credential payload is a JSON object with the following **OPTIONAL** fields:

```
{
  "line1": "123 Main St",
  "line2": "Apt 4B",
  "city": "San Francisco",
  "state": "CA",
  "zip": "94102",
  "countryCode": "US"
}
```

Field	Type	Required	Description
line1	string	OPTIONAL	Street address, line 1.
line2	string	OPTIONAL	Street address, line 2.
city	string	OPTIONAL	City or locality.
state	string	OPTIONAL	State, province, or region.
zip	string	OPTIONAL	Postal or ZIP code.
countryCode	string	OPTIONAL	ISO 3166-1 alpha-2 country code (e.g., "US").

Table 7

All billing fields are **OPTIONAL** within the billing object. Client Enablers **SHOULD** include whichever fields are available from the cardholder's stored billing profile.

If `billingRequired` is true but the credential omits the billing field, the server **MAY** reject the credential or proceed at its discretion.

8. Verification Procedure

Servers **MUST** verify Payment credentials per the verification requirements in [\[I-D.payment-intent-charge\]](#). The following procedure implements those requirements for the card method:

1. Decode the credential: base64url-decode the token from the Authorization: Payment header and parse as JSON.

2. Verify challenge binding: confirm `challenge.id` matches an outstanding challenge issued by this server.
3. Verify the challenge has not expired (check the `expires` field).
4. Verify the method: confirm `challenge.method` equals "card".
5. Verify network acceptance: confirm `payload.network` is in the `acceptedNetworks` list from `methodDetails`.
6. Reject replays: confirm this `challenge.id` has not been previously fulfilled. Mark it as consumed.
7. Verify payment amount: the Server Enabler **MUST** confirm the authorization amount matches the amount from the request object.

8.1. Challenge Binding

Servers **MUST** verify that the credential corresponds to the exact challenge issued. Challenge IDs **SHOULD** be cryptographically bound (e.g., HMAC) to their parameters to enable stateless verification. Bound parameters include:

- Challenge ID
- Amount and currency (from the request object)
- Accepted networks
- Recipient (merchant ID)
- Realm
- Expiry timestamp
- Encryption key identifier (`kid`)

Alternatively, servers **MAY** store challenge parameters server-side and verify by lookup using the challenge ID.

8.2. Idempotency and Replay Protection

Per [I-D.payment-intent-charge], each credential **MUST** be usable only once per challenge. Servers **MUST** reject replayed credentials.

Servers **MUST** use `challenge.id` as an idempotency key when forwarding to the Server Enabler. This prevents duplicate charges from retried requests.

Replay behavior:

- Same `challenge.id`, credential already successfully processed: server **MUST** return the cached 200 response with `Payment-Receipt`.
- Same `challenge.id`, prior processing failed: server **MUST** return HTTP 409 Conflict.
- Same `challenge.id`, challenge expired: server **MUST** return HTTP 402 with a fresh challenge.
- Challenge IDs **MUST** contain at least 128 bits of entropy.

- Servers **MUST** store challenge state with a TTL of at least the challenge expiry window plus a grace period (**RECOMMENDED**: expiry + 5 minutes).
- Servers **SHOULD** purge challenge state after the TTL expires.

9. Settlement Procedure

After credential verification, the Server Enabler decrypts the network token using the private key corresponding to the encryption key published in the challenge, and submits an authorization request to the card network.

On approval, the server returns HTTP 200 with a Payment-Receipt header and the requested resource. Servers **SHOULD** return 200 immediately after authorization approval, even though final fund settlement is pending.

If the issuing bank declines, the server **MUST** return an error response and **SHOULD** issue a fresh 402 challenge to allow the client to retry.

9.1. Receipt Generation

Upon successful authorization, servers **MUST** return a Payment-Receipt header per [I-D.httpauth-payment]. Servers **MUST NOT** include Payment-Receipt on error responses.

Decoded receipt:

```
{
  "challengeId": "ch_9xK2mR4vB7nQ",
  "method": "card",
  "status": "success",
  "reference": "visa_txn_abc123",
  "timestamp": "2026-02-19T12:05:30Z",
  "externalId": "order_12345"
}
```

Receipt fields:

Field	Type	Required	Description
challengeId	string	REQUIRED	The challenge ID that was fulfilled.
method	string	REQUIRED	"card".
status	string	REQUIRED	"success".
reference	string	REQUIRED	Authorization reference from the Server Enabler.
timestamp	string	REQUIRED	[RFC3339] timestamp of the authorization.

Field	Type	Required	Description
externalId	string	OPTIONAL	Echo of the externalId from the request, if provided.

Table 8

10. Security Considerations

10.1. Transport Security

All MPP exchanges **MUST** occur over TLS 1.2 or higher (TLS 1.3 recommended). Plain HTTP **MUST** be rejected. Clients **SHOULD** verify the server's TLS certificate.

10.2. Credential Security

The `encryptedPayload` field is a JWE [RFC7516] compact token not readable by the client (Section 7.3). The Client Enabler encrypts the token payload using JWE with the server's encryption key (Section 6.3), provided as a JWK via `encryptionJwk` or resolved from `jwtUri` in `methodDetails`. The JWE provides both confidentiality (RSA-OAEP-256 key wrapping + AES-256-GCM content encryption) and integrity (GCM authentication tag). Only the Server Enabler holding the corresponding private key can decrypt and process it.

Only encrypted network tokens travel in the credential. The client never has access to decrypted token material.

10.3. Billing Data Handling

Billing information (Section 7.6), `cardholderFullName`, and `paymentAccountReference` are personally identifiable information (PII) and **SHOULD** be handled in accordance with applicable privacy regulations (e.g., GDPR, CCPA).

- Billing data, `cardholderFullName`, and `paymentAccountReference` are transmitted as plaintext within the credential payload, protected by TLS in transit (Section 10.1).
- `paymentAccountReference` (PAR) is a persistent cross-channel identifier that can correlate a cardholder across merchants and payment methods. Servers **SHOULD** treat PAR with the same care as billing data and **SHOULD NOT** use it for cross-merchant tracking beyond the server's own business relationship with the cardholder.
- Servers and intermediaries **SHOULD NOT** log billing data, `cardholderFullName`, or `paymentAccountReference` in plaintext unless required for order fulfillment or dispute resolution.
- Servers **SHOULD** retain billing data only as long as needed for their business purpose (address verification, fulfillment, dispute resolution) and in accordance with applicable privacy regulations.

11. IANA Considerations

11.1. Payment Method Registration

This document registers the following payment method in the "HTTP Payment Methods" registry established by [I-D.httpauth-payment]:

Method Identifier	Description	Reference
card	Card payment via encrypted network token credential	This document

Table 9

Contact: Visa (jbrans@visa.com)

11.2. Payment Intent Registration

This document registers the following payment intent in the "HTTP Payment Intents" registry established by [I-D.httpauth-payment]:

Intent	Applicable Methods	Description	Reference
charge	card	One-time card payment via encrypted network token	This document

Table 10

12. References

12.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <<https://www.rfc-editor.org/info/rfc3339>>.
- [RFC7516] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, DOI 10.17487/RFC7516, May 2015, <<https://www.rfc-editor.org/info/rfc7516>>.
- [RFC7517] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <<https://www.rfc-editor.org/info/rfc7517>>.

- [RFC7518] Jones, M., "JSON Web Algorithms (JWA)", RFC 7518, DOI 10.17487/RFC7518, May 2015, <<https://www.rfc-editor.org/info/rfc7518>>.
- [RFC8017] Moriarty, K., Ed., Kaliski, B., Jonsson, J., and A. Rusch, "PKCS #1: RSA Cryptography Specifications Version 2.2", RFC 8017, DOI 10.17487/RFC8017, November 2016, <<https://www.rfc-editor.org/info/rfc8017>>.
- [RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <<https://www.rfc-editor.org/info/rfc4648>>.
- [RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <<https://www.rfc-editor.org/info/rfc8259>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC9110] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <<https://www.rfc-editor.org/info/rfc9110>>.
- [I-D.httpauth-payment] Moxey, J., "The 'Payment' HTTP Authentication Scheme", January 2026, <<https://datatracker.ietf.org/doc/draft-ryan-httpauth-payment/>>.
- [I-D.payment-intent-charge] Moxey, J., "Charge Intent for HTTP Payment Authentication", March 2026, <<https://datatracker.ietf.org/doc/draft-payment-intent-charge/>>.

12.2. Informative References

- [EMV-SRC-API] EMVCo, LLC, "EMV Secure Remote Commerce - API Specification", October 2025, <<https://www.emvco.com/emv-technologies/secure-remote-commerce/>>.
- [VISA-INTELLIGENT-COMMERCE] Visa Inc., "Visa Intelligent Commerce", n.d., <<https://developer.visa.com/capabilities/visa-intelligent-commerce>>.
- [VISA-TAP] Visa Inc., "Visa Trusted Agent Protocol", n.d., <<https://developer.visa.com/capabilities/trusted-agent-protocol/>>.

Appendix A. Client Enabler Profile

This appendix describes an illustrative HTTP interface for Client Enablers (CEs). A Client Enabler is any entity that accepts challenge context from a client, provisions a network token and cryptogram, encrypts the result using the server's encryption key (Section 6.3), and returns the credential payload. Client Enablers include vault providers, token service providers, and PSPs acting as issuers.

The interface described here is informative. Endpoint URLs, paths, and authentication mechanisms are CE-defined.

A.1. Token Request Interface

The client sends a request to the Client Enabler with the card identifier and the full challenge context (including the encryption key from methodDetails as encryptionJwk or jwksUri + kid). The Bearer token shown below is illustrative.

Request:

```
POST /v1/payment-tokens HTTP/1.1
Host: api.vault-provider.com
Content-Type: application/json
Authorization: Bearer <agent_api_key>
```

```
{
  "cardId": "card_abc123",
  "challenge": {
    "id": "ch_9xK2mR4vB7nQ",
    "realm": "api.merchant.com",
    "amount": "4999",
    "currency": "usd",
    "acceptedNetworks": ["visa", "mastercard"],
    "merchantName": "Acme Corp",
    "encryptionJwk": {
      "kty": "RSA",
      "kid": "enc-2026-01",
      "use": "enc",
      "alg": "RSA-OAEP-256",
      "n": "0vx7agoebGcQSuu...",
      "e": "AQAB"
    }
  }
}
```

A conforming Client Enabler is expected to:

- Validate that the cardId exists and belongs to the authenticated client.
- Provision a network token and cryptogram for the transaction.
- Resolve the encryption key per the procedure in [Section 6.3](#): use encryptionJwk directly, or fetch jwksUri and select by kid.
- Validate the key: kty is "RSA", alg is "RSA-OAEP-256", use is "enc".
- Encrypt the token payload as a JWE [[RFC7516](#)] compact serialization using the resolved key, with alg set to "RSA-OAEP-256" and enc set to "A256GCM" ([Section 7.3](#)).
- Return the encrypted token along with display metadata and authentication context.

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
```

```
{
  "encryptedPayload": "<JWE compact serialization>",
  "network": "visa",
  "panLastFour": "4242",
  "panExpirationMonth": "06",
  "panExpirationYear": "2028",
  "cardholderFullName": "Jane Smith",
  "paymentAccountReference": "PAR9876543210987654321012345"
}
```

The response body contains the credential payload fields defined in [Section 7.2](#). The client includes these fields directly in the credential payload.

Appendix B. ABNF Collected

This appendix imports `quoted-string`, `token`, and `base64url` from [\[I-D.httpauth-payment\]](#).

```
card-challenge = "Payment" 1*SP
  "id=" quoted-string ","
  "realm=" quoted-string ","
  "method=" DQUOTE "card" DQUOTE ","
  "intent=" DQUOTE "charge" DQUOTE ","
  "request=" base64url
  *("," 1*SP token "=" ( token / quoted-string ))

card-credential = "Payment" 1*SP base64url
```

Appendix C. Examples

C.1. Charge Example (HTTP Transport)

Step 1: Client requests resource

```
GET /api/data HTTP/1.1
Host: api.merchant.com
```

Step 2: Server issues payment challenge

```
HTTP/1.1 402 Payment Required
WWW-Authenticate: Payment
  id="ch_9xK2mR4vB7nQ",
  realm="api.merchant.com",
  method="card",
  intent="charge",
  expires="2026-02-19T12:10:00Z",
  request="eyJhbW91bnQiOiI0OTk5Iiwia3VycmVuY3kiOiJlc2QiLCJyZWNP
    cGllbnQiOiJtZXJjaF9hYmMxMjMiLCJkZXNjcmlwdGlubiI6IiBybyBw
    bGFuI0KAlCBtb250aGx5IHN1YnNjcm1wdGlubiIsImV4dGVybmFsSWQi
    OiJvcmlcl18xMjM0NSIsIm1ldGhvZERldGFpbHM0OnsiYWNjZXB0ZWRO
    ZXR3b3JrcyI6WyJ2aXNhIiwibWFzdGVyY2FyZCI6ImFtZXgiXSibWVy
    Y2hhbnR0YW11IjoIQWntZSBDb3JwIiwiaWlscGluc2V0IjoiYWNjZXB0
    cnVlLCJlbnNyeXB0aW9uSndrIjp7Imt0eSI6IiJ1JTQSI6ImV4dGVy
    Yy0yMDI2LTAxIiwidXN1IjojZjIiwiaWw3IjojIiwiaWw3IjojIiwiaWw3
    Iiwib2E2dDdhZ291YkdjUVN1dS4uLiIsImUiOiJBUUF0In19fQ"
Cache-Control: no-store
Content-Type: application/problem+json
```

```
{
  "type": "https://paymentauth.org/problems/payment-required",
  "title": "Payment Required",
  "status": 402,
  "detail": "This resource requires payment"
}
```

Decoded request:

```
{
  "amount": "4999",
  "currency": "usd",
  "recipient": "merch_abc123",
  "description": "Pro plan – monthly subscription",
  "externalId": "order_12345",
  "methodDetails": {
    "acceptedNetworks": ["visa", "mastercard", "amex"],
    "merchantName": "Acme Corp",
    "billingRequired": true,
    "encryptionJwk": {
      "kty": "RSA",
      "kid": "enc-2026-01",
      "use": "enc",
      "alg": "RSA-OAEP-256",
      "n": "0vx7agoebGcQSuu...",
      "e": "AQAB"
    }
  }
}
```

Step 3: Client obtains credential from Client Enabler

The client sends challenge parameters (including encryption JWK from methodDetails) to its Client Enabler ([Appendix A](#)). The CE provisions a network token and cryptogram via existing TSPs, encrypts the token with the server's encryption key, and returns the result.

Step 4: Client retries request with credential

```
GET /api/data HTTP/1.1
Host: api.merchant.com
Authorization: Payment eyJjaGFsbGVuZ2UiOonsiaWQiOiJjaF85eEsyb
VI0dkI3blEiLCJyZWZfSbSI6ImFwaS5tZXJjaGFudC5jb20iLCJtZXRob2QiO
iJjYXJkIiwiaW50ZW50IjoIY2hhcmdlIiwicmVxdWVzdCI6ImV5SmhiVzkyY
m5RaU9pSTBPVGs1SWk0dUxuMCIsImV4cGlyZXMiOiIyMDI2LTAyLTE5VDEyO
jEwOjAwWiJ9LCJwYXlsb2FkIjpb7ImVuY3J5cHRlZFBheWxvYWQiOiJleUpoY
kdjaU9pS1NVMEV0Li4uPEpXRSBjb21wYWN0Pi4uLlhGQm9NWVAb2RldFpkd
lRrRnZTa1EiLCJuzXR3b3JrIjoIdmIzYSIsInBhbKxhc3RGb3VyIjoI0NDI0M
iIsInBhbKv4cGlyYXRpb25Nb250aCI6IjA2IiwicGFuRXhwaXJhdGlvblllY
XIiOiIyMDI4IiwiaW50ZW50IjoIY2hhcmdlIiwicmVxdWVzdCI6ImV5SmhiVzkyY
3VudHJ5Q29kZSI6I1VTIn0sImNhcmRob2xkZXJGdWxsTmFtZSI6IkpPhmUgU
21pdGgiLCJwYXltZW50QWNjb3VudFJlZmVyZW5jZSI6I1BBUjk4NzY1NDMyM
TA5ODc2NTQzMjEwMTIzNDUifX0
```

Decoded credential:

```
{
  "challenge": {
    "id": "ch_9xK2mR4vB7nQ",
    "realm": "api.merchant.com",
    "method": "card",
    "intent": "charge",
    "request": "eyJhbW91bnQiOiI00Tk5Ii...",
    "expires": "2026-02-19T12:10:00Z"
  },
  "payload": {
    "encryptedPayload": "<base64-encoded RSA-OAEP ciphertext>",
    "network": "visa",
    "panLastFour": "4242",
    "panExpirationMonth": "06",
    "panExpirationYear": "2028",
    "billingAddress": {
      "zip": "94102",
      "countryCode": "US"
    },
    "cardholderFullName": "Jane Smith",
    "paymentAccountReference": "PAR9876543210987654321012345"
  }
}
```

Step 5: Server processes payment and returns resource

```
HTTP/1.1 200 OK
Payment-Receipt: eyJjaGFsbGVuZ2VJZCI6ImNoXzI4SzMtUjR2QjduUSIs
  Im1ldGhvZCI6ImNhcmQiLCJzdGF0dXMiOiJzdWNjZXNzIiwicmVmZXJl
  bmNlIjoiaWwzYV90eG5fYWJjMTIzIiwidGltZXN0YW1wIjoiaWwzYV90eG5f
  Mi0xOVQxMjowNTozMFoiLCJleHRlcm5hbElkIjoib3JkZXJfMTIzNDUi
  fQ
Cache-Control: private
Content-Type: application/json
```

```
{
  "data": "Here is your requested resource..."
}
```

Decoded receipt:

```
{
  "challengeId": "ch_9xK2mR4vB7nQ",
  "method": "card",
  "status": "success",
  "reference": "visa_txn_abc123",
  "timestamp": "2026-02-19T12:05:30Z",
  "externalId": "order_12345"
}
```

Appendix D. Acknowledgements

The authors thank Visa's acceptance and tokenization partners for their contributions to the card payment ecosystem that informed this specification. The authors also thank Brendan Ryan for his review of this document.

Author's Address

Jacob Brans

Visa

Email: jbrans@visa.com