
Workgroup: Network Working Group
Internet-Draft: draft-stripe-charge-00
Published: 9 September 2026
Intended Status: Informational
Expires: 13 March 2027
Authors: B. Ryan S. Kaliski
Tempo Labs Stripe

Stripe Charge Intent for HTTP Payment Authentication

Abstract

This document defines the "charge" intent for the Stripe payment method within the Payment HTTP Authentication Scheme [I-D.httpauth-payment]. It specifies how clients and servers exchange one-time payments using Shared Payment Tokens (SPTs).

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 13 March 2027.

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. Stripe Charge Flow	3
1.2. Relationship to the Payment Scheme	4
2. Requirements Language	4
3. Terminology	4
4. Intent Identifier	5
5. Intent: "charge"	5
6. Request Schema	5
6.1. Shared Fields	5
6.2. Method Details	6
7. Credential Schema	7
8. Verification Procedure	7
8.1. Challenge Binding	8
9. Settlement Procedure	8
9.1. Stripe Connect Settlement	9
9.2. Receipt Generation	10
10. Security Considerations	11
10.1. SPT Single-Use Constraint	11
10.2. Amount Verification	11
10.3. Stripe Connect Parameter Integrity	11
10.4. PCI DSS Compliance	12
10.5. HTTPS Requirement	12
11. IANA Considerations	12
11.1. Payment Intent Registration	12
12. References	12
12.1. Normative References	12

12.2. Informative References	13
Appendix A. ABNF Collected	13
Appendix B. Examples	13
B.1. Charge Example (HTTP Transport)	13
Appendix C. Acknowledgements	15
Authors' Addresses	15

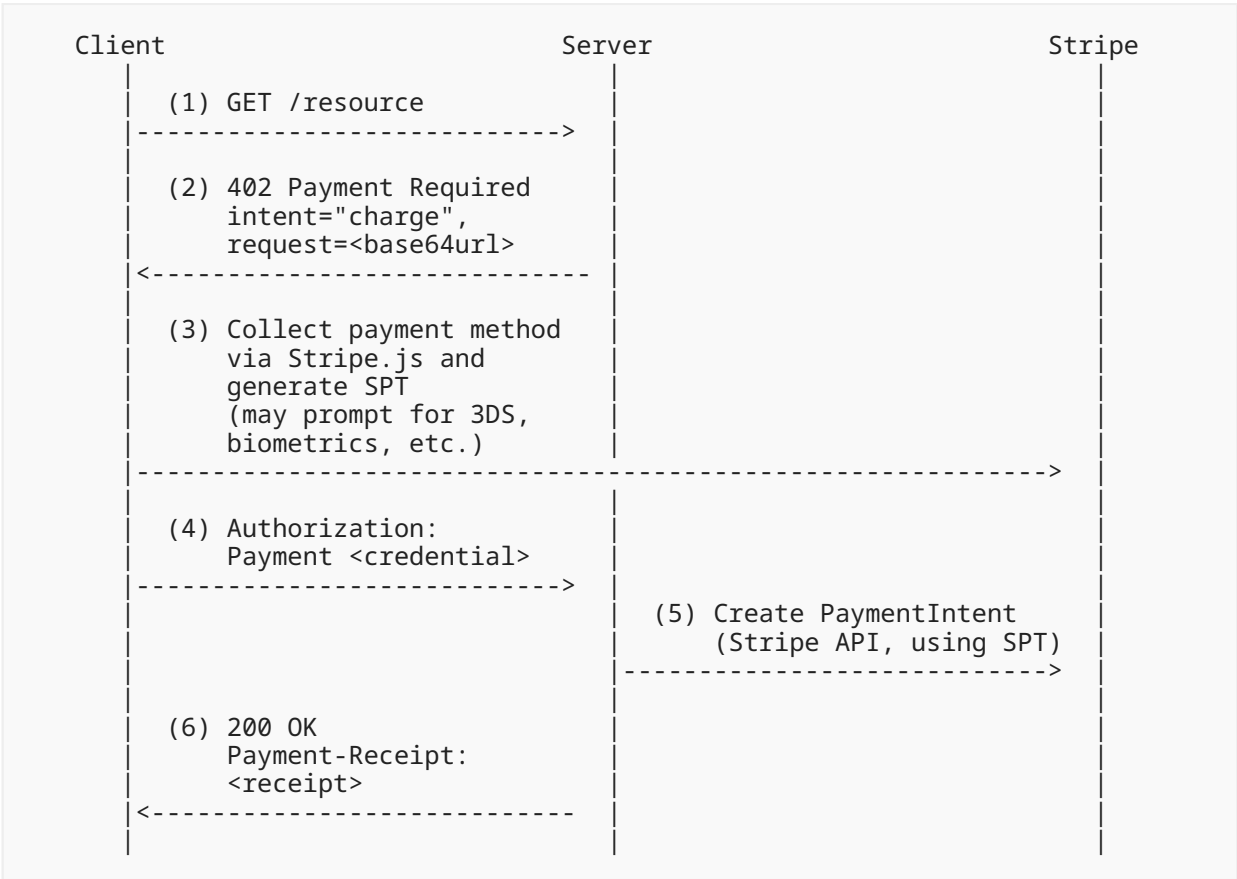
1. Introduction

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

Stripe provides payment processing through SPTs, which are single-use tokens that represent payment authorization. SPTs abstract away the complexity of payment method details (cards, bank accounts, wallets) and provide a unified interface for payment acceptance.

1.1. Stripe Charge Flow

The following diagram illustrates the Stripe charge payment flow:



1.2. Relationship to the Payment Scheme

This document is a payment method intent specification as defined in [I-D.httpauth-payment]. It defines the request and payload structures for the charge intent of the stripe payment method, along with verification and settlement procedures.

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

Shared Payment Token (SPT)

A single-use token (prefixed with `spt_`) that represents authorization to charge a payment method. SPTs are created by clients using the Stripe API and consumed by servers to process payments. Both the Client and Server require a Stripe account. In the Stripe API, SPTs are referenced as `shared_payment_granted_token` on `PaymentIntent` creation. See [STRIDE-SPT].

Business Network Profile A Stripe profile is a business's public identity on Stripe. With a Stripe profile, businesses can find, verify, and connect with each other on Stripe. Learn more: <https://docs.stripe.com/get-started/account/profile>

Payment Intent A Stripe API object that tracks the lifecycle of a customer payment, from creation through settlement. Not to be confused with the HTTP Payment Auth protocol's "payment intent" parameter. Learn more: <https://docs.stripe.com/payments/payment-intents>

4. Intent Identifier

This specification defines the following intent for the `stripe` payment method:

```
charge
```

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

5. Intent: "charge"

A one-time payment of the specified amount. The server processes the payment immediately upon receiving the SPT.

Fulfillment mechanism:

1. **Shared Payment Token (SPT):** The payer creates an SPT using the Stripe API, which the server uses to create a `PaymentIntent` via Stripe.

6. Request Schema

The `request` parameter in the `WWW-Authenticate` challenge contains a base64url-encoded JSON object with the following fields. The JSON **MUST** be serialized using JSON Canonicalization Scheme (JCS) [RFC8785] before base64url encoding, per [I-D.httpauth-payment].

6.1. Shared Fields

Field	Type	Required	Description
amount	string	REQUIRED	Amount in smallest currency unit (e.g., cents), encoded as a string
currency	string	REQUIRED	Three-letter ISO currency code (e.g., "usd")

Field	Type	Required	Description
description	string	OPTIONAL	Human-readable payment description
externalId	string	OPTIONAL	Merchant's identifier (e.g., order ID, cart ID)
recipient	string	OPTIONAL	Payment recipient identifier

Table 1

6.2. Method Details

Field	Type	Required	Description
methodDetails.networkId	string	REQUIRED	Stripe Business Network Profile ID
methodDetails.paymentMethodTypes	[]string	REQUIRED	The list of payment method types that the seller can process.
methodDetails.metadata	object	OPTIONAL	Key-value pairs for additional context

Table 2

Example:

```
{
  "amount": "5000",
  "currency": "usd",
  "description": "Premium API access for 1 month",
  "externalId": "order_12345",
  "methodDetails": {
    "networkId": "profile_1MqDcVKA5fE02tZvKQm9g8Yj",
    "paymentMethodTypes": ["card", "link"]
  }
}
```

The client fulfills this by creating an SPT using Stripe:

```
const spt = await stripe.sharedPayment.issuedTokens.create({
  payment_method: 'pm_123',
  usage_limits: {
    currency: 'usd',
    max_amount: 5000,
    expires_at: Timestamp
  },
  seller_details: {
    networkId: 'profile_123'
  }
});
// Returns: { id: 'spt_1N...' }
```

7. Credential Schema

The Payment credential is a base64url-encoded JSON object containing challenge and payload fields per [I-D.httpauth-payment]. For Stripe charge, the payload object contains the following fields:

Field	Type	Required	Description
spt	string	REQUIRED	Shared Payment Token ID (starts with spt_)
externalId	string	OPTIONAL	Client's reference ID

Table 3

Example:

```
{
  "spt": "spt_1N4Zv32eZvKYlo2CPhVPkJlW",
  "externalId": "client_order_789"
}
```

8. Verification Procedure

Servers **MUST** verify Payment credentials for charge intent:

1. Verify the challenge ID matches the one issued
2. Verify the challenge has not expired
3. Extract the spt from the credential payload
4. Verify the SPT has not been previously used (replay protection)
5. Validate the SPT exists and is valid via Stripe API (optional pre-check)

Servers **MUST** complete challenge ID validation and expiry checks (steps 1-2) before processing credential material (steps 3-5). This ensures basic request validity is established before accessing payment tokens.

8.1. Challenge Binding

Servers **MUST** verify that the credential corresponds to the exact challenge issued. This includes validating:

- Challenge ID
- Amount (if specified in request)
- Currency
- Business Network (if specified)
- Any custom metadata

9. Settlement Procedure

Synchronous settlement:

1. Server receives and verifies the credential ([Section 8](#))
2. Server creates a Stripe PaymentIntent with `confirm: true` and the SPT as `shared_payment_granted_token`. The server **MAY** include Stripe Connect settlement parameters as described in [Section 9.1](#):

```
const stripeDetails = request.methodDetails || {};  
const settlementPolicy = getServerSettlementPolicy(challenge, request);  
  
const paymentIntentParams = {  
  amount: Number(request.amount),  
  currency: request.currency,  
  shared_payment_granted_token: credential.spt,  
  confirm: true,  
  automatic_payment_methods: {  
    enabled: true,  
    allow_redirects: 'never'  
  },  
  metadata: {  
    ...(stripeDetails.metadata || {}),  
    challenge_id: challenge.id  
  }  
};  
  
if (settlementPolicy.applicationFeeAmount !== undefined) {  
  paymentIntentParams.application_fee_amount =  
    settlementPolicy.applicationFeeAmount;  
}  
  
if (settlementPolicy.onBehalfOf !== undefined) {  
  paymentIntentParams.on_behalf_of = settlementPolicy.onBehalfOf;  
}  
  
if (settlementPolicy.transferData !== undefined) {  
  paymentIntentParams.transfer_data = {  
    destination: settlementPolicy.transferData.destination
```

```
};

if (settlementPolicy.transferData.amount !== undefined) {
  paymentIntentParams.transfer_data.amount =
    settlementPolicy.transferData.amount;
}

if (settlementPolicy.transferGroup !== undefined) {
  paymentIntentParams.transfer_group = settlementPolicy.transferGroup;
}

const paymentIntentOptions = {
  idempotencyKey: `${challenge.id}_${credential.spt}`
};

if (settlementPolicy.stripeAccount !== undefined) {
  paymentIntentOptions.stripeAccount = settlementPolicy.stripeAccount;
}

const paymentIntent = await stripe.paymentIntents.create(
  paymentIntentParams,
  paymentIntentOptions
);
```

1. Server **MUST** verify the PaymentIntent status is "succeeded" before returning 200 with Payment-Receipt header
2. If the PaymentIntent fails or requires additional action, server returns 402 with a new challenge

Idempotency:

Servers **SHOULD** include an idempotency key derived from the challenge ID and SPT when creating PaymentIntents. This prevents duplicate charges if the client retries a request.

Settlement timing:

Stripe processes fund transfers asynchronously. Servers **SHOULD** return 200 immediately after PaymentIntent confirmation (status "succeeded"), even if final fund settlement to the merchant is pending.

9.1. Stripe Connect Settlement

Servers that use Stripe Connect **MAY** apply Connect parameters when creating the Stripe PaymentIntent. These parameters are settlement policy inputs controlled by the server, not payment request fields that clients need in order to create an SPT.

The following Stripe PaymentIntent create parameters are compatible with this specification:

Stripe parameter	Description
Stripe-Account header or SDK stripeAccount option	Connected account used as the Stripe account context for the request
application_fee_amount	Platform application fee amount in the smallest currency unit
on_behalf_of	Connected account used as the business of record
transfer_data[destination]	Connected account that receives transferred funds
transfer_data[amount]	Amount transferred to transfer_data[destination]
transfer_group	Reconciliation token linking related charges and transfers

Table 4

Servers **SHOULD** derive Connect settlement parameters from server-side merchant configuration, platform policy, order state, or equivalent trusted state. Servers **MUST NOT** include Connect settlement parameters in the MPP challenge.

9.2. Receipt Generation

Upon successful settlement, servers **MUST** return a Payment-Receipt header per [I-D.httpauth-payment]. Servers **MUST NOT** include a Payment-Receipt header on error responses; failures are communicated via HTTP status codes and Problem Details.

The receipt payload for Stripe charge:

Field	Type	Description
method	string	"stripe"
reference	string	Stripe PaymentIntent ID (e.g., "pi_1N4...")
status	string	"success"
timestamp	string	[RFC3339] confirmation time
externalId	string	OPTIONAL . Echoed from credential payload

Table 5

10. Security Considerations

10.1. SPT Single-Use Constraint

SPTs are single-use tokens. Stripe automatically prevents SPT reuse at the API level, and idempotency keys ([Section 9](#)) prevent duplicate PaymentIntent creation. Servers **MUST** enforce single-use challenge IDs per [\[I-D.httpauth-payment\]](#) and **SHOULD** use Stripe idempotency keys to prevent repeated charges. Servers **MAY** additionally maintain a local replay cache of consumed challenge IDs.

10.2. Amount Verification

Clients **MUST** verify the payment amount in the challenge matches their expectation before creating an SPT. The SPT usage limits constrain the currency, maximum amount, and expiration window granted to the seller, but those limits are derived from the challenge parameters the client accepts.

Verification checklist:

1. Verify the amount matches the expected cost
2. Verify the currency matches the expected currency
3. Verify the description matches the expected service
4. Verify the challenge hasn't expired
5. Verify the server's identity (TLS certificate validation)

10.3. Stripe Connect Parameter Integrity

Servers that use Stripe Connect settlement parameters **MUST** validate those parameters before creating the PaymentIntent:

- Connected account identifiers, including `stripeAccount`, `on_behalf_of`, and `transfer_data[destination]`, **MUST** refer to connected accounts that the server is authorized to use for the current request.
- `transfer_data[destination]` **MUST** be present when `transfer_data` is present.
- `application_fee_amount`, when present, **MUST** be a non-negative integer no greater than the PaymentIntent amount.
- `transfer_data[amount]`, when present, **MUST** be a non-negative integer no greater than the PaymentIntent amount.
- `transfer_group`, when present, **MUST** be derived from an order, invoice, request, or similar reconciliation identifier and **MUST NOT** be treated as secret.

Servers **MUST NOT** accept Connect settlement parameters from clients unless an explicit trust boundary authorizes that client to control settlement routing or platform fees.

10.4. PCI DSS Compliance

Stripe's SPT model ensures clients never handle raw payment method details, significantly reducing PCI DSS compliance scope.

10.5. HTTPS Requirement

All communication **MUST** use TLS 1.2 or higher. Shared Payment Tokens **MUST** only be transmitted over HTTPS connections.

11. IANA Considerations

11.1. Payment Intent Registration

This specification registers the "charge" intent for the "stripe" payment method in the Payment Intent Registry established by [I-D.httpauth-payment]:

- **Intent:** charge
- **Method:** stripe
- **Specification:** [this document]

Contact: Stripe (stevekaliski@stripe.com) and Tempo Labs (brendan@tempo.xyz)

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>>.
- [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>>.
- [RFC8785] Rundgren, A., Jordan, B., and S. Erdtman, "JSON Canonicalization Scheme (JCS)", RFC 8785, DOI 10.17487/RFC8785, June 2020, <<https://www.rfc-editor.org/info/rfc8785>>.
- [RFC7235] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Authentication", RFC 7235, DOI 10.17487/RFC7235, June 2014, <<https://www.rfc-editor.org/info/rfc7235>>.

[I-D.httpauth-payment] Moxey, J., "The 'Payment' HTTP Authentication Scheme", January 2026, <<https://datatracker.ietf.org/doc/draft-ryan-httpauth-payment/>>.

12.2. Informative References

[STRIPE-API] Stripe, Inc., "Stripe API Reference", n.d., <<https://stripe.com/docs/api>>.

[STRIPE-SPT] Stripe, Inc., "Shared payment tokens", n.d., <<https://docs.stripe.com/agent-commerce/concepts/shared-payment-tokens>>.

Appendix A. ABNF Collected

```
stripe-charge-challenge = "Payment" 1*SP
    "id=" quoted-string ","
    "realm=" quoted-string ","
    "method=" DQUOTE "stripe" DQUOTE ","
    "intent=" DQUOTE "charge" DQUOTE ","
    "request=" base64url-nopad

stripe-charge-credential = "Payment" 1*SP base64url-nopad

; Base64url encoding without padding per RFC 4648 Section 5
base64url-nopad = 1*( ALPHA / DIGIT / "-" / "_" )
```

Appendix B. Examples

B.1. Charge Example (HTTP Transport)

Step 1: Client requests resource

```
GET /api/generate HTTP/1.1
Host: api.example.com
```

Step 2: Server issues payment challenge

```
HTTP/1.1 402 Payment Required
WWW-Authenticate: Payment id="ch_1a2b3c4d5e",
  realm="api.example.com",
  method="stripe",
  intent="charge",

request="eyJhbW91bnQiOiI1MDAwIiwiaY3VycmVuY3kiOiJ1c2QiLCJkZXNjcmlwdGlvbiI6IkFJIGdldmVhYXRpb24ifQ"
Cache-Control: no-store
Content-Type: application/json

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

Decoded request: ~~~ json { "amount": "5000", "currency": "usd", "description": "AI generation" }
~~~

### Step 3: Client creates SPT and submits credential

```
GET /api/generate HTTP/1.1
Host: api.example.com
Authorization: Payment
eyJjaGFsbGVuZ2UiOnsiaWQiOiJjaF8xYTJiM2M0ZDVlIiwicmVhbmG0iOiJhcGkuZXhhbXBsZS5jb20iLCJtZXRob2QiOiJzdHJpcGUlLCJpbmRlbnQiOiJjaGFyZ2UiLCJyZXF1ZXN0IjoizXlKaGJXOTFib1FpT2lJMU1EQXdJaXdpWTNweWNTVnVZM2tpT2lKMWMyUWlMQ0prWlh0amNtbHdkR2x2Ym1JNk1rRkpJR2RsYm1WeVlYUnBiMjRpb2EiLCJleHBpcmVzIjoimjAyNS0wMS0xNVQxMjowNTowMFoifSwicGF5bG9hZCI6eyJzchQiOiJzcHRfMU40WnYzMmVadktZbG8yQ1BoVlBrSmxXIn19
```

Decoded credential: ~~~ json { "challenge": { "id": "ch\_1a2b3c4d5e", "realm": "api.example.com", "method": "stripe", "intent": "charge", "request": "eyJhbW91bnQiOiI1MDAwIiwiaY3VycmVuY3kiOiJ1c2QiLCJkZXNjcmlwdGlvbiI6IkFJIGdldmVhYXRpb24ifQ", "expires": "2025-01-15T12:05:00Z" }, "payload": { "spt": "spt\_1N4Zv32eZvKYlo2CPhVPkjlW" } } ~~~

### Step 4: Server processes payment and returns resource

```
HTTP/1.1 200 OK
Payment-Receipt:
eyJtZXRob2QiOiJzdHJpcGUlLCJyZWZlcmVuY2UiOiJwaV8xTjRadjMyZVp2S1lsbzJDUGhwUGtKbFciLCJzdGF0dXMiOiJzdWNjZXNzIiwidGltZXN0YW1wIjoimjAyNS0wMS0xNVQxMjowNDowMloifQ
Cache-Control: private
Content-Type: text/plain

Here is your generated content...
```

Decoded receipt: ~~~ json { "method": "stripe", "reference": "pi\_1N4Zv32eZvKYlo2CPhVPkJlW",  
"status": "success", "timestamp": "2025-01-15T12:04:32Z" } ~~~

## Appendix C. Acknowledgements

The authors thank the Tempo community for their feedback on this specification.

### Authors' Addresses

**Brendan Ryan**

Tempo Labs

Email: [brendan@tempo.xyz](mailto:brendan@tempo.xyz)

**Steve Kaliski**

Stripe

Email: [stevekaliski@stripe.com](mailto:stevekaliski@stripe.com)